

**SFB 1053**



# Mechanism Transitions: A New Paradigm for a Highly Adaptive Internet

*Authors\*:*

Alexander Frömmgen, Mohamed Hassan, Roland Kluge,  
Mahdi Mousavi, Max Mühlhäuser, Sabrina Müller,  
Mathias Schnee, Michael Stein, Markus Weckesser

*Editor:*

Max Mühlhäuser

Project Group *Transition Methodology*  
Collaborative Research Center MAKI -  
Multi-Mechanisms Adaptation for the Future Internet  
Technische Universität Darmstadt

March 24, 2016

---

\*All authors in alphabetical order

## Contents

<b>1</b>	<b>Mechanism Transitions: A New Paradigm</b>	<b>5</b>
1.1	Resulting Future Internet Challenge . . . . .	5
<b>2</b>	<b>From OSI via Internet to MAKI</b>	<b>7</b>
2.1	The OSI Model as a Starting Point . . . . .	7
2.2	The Internet in Relation to OSI . . . . .	8
2.3	Towards the MAKI Internet Model . . . . .	10
<b>3</b>	<b>Modeling MAKI Mechanism Transitions</b>	<b>10</b>
3.1	Requirements Imposed by the Reality of Today's Internet . . . .	10
3.2	Requirements Imposed by Mechanism Transitions . . . . .	11
<b>4</b>	<b>The MAKI Reference Model</b>	<b>15</b>
4.1	The Structural Model . . . . .	15
4.1.1	Basic Reference Model for Mechanisms . . . . .	16
4.1.2	Possible Relationships of Coexisting Mechanisms . . .	18
4.2	Behavioral Model: Transitions . . . . .	21
4.3	Behavioral Model: Transition Decision . . . . .	24
<b>5</b>	<b>MAKI Transitions: Structural Aspects</b>	<b>28</b>
5.1	Structural Modeling Based on Metamodels . . . . .	29
5.1.1	Definitions . . . . .	29
5.1.2	Example 1: Wireless Sensor Networks . . . . .	29
5.1.3	Example 2: Location-Assisted Publish-Subscribe . . . .	31
5.1.4	Example 3: Adaptive Video Streaming System . . . . .	33
5.2	Describing System Configurations using Feature Models . . . .	34
5.2.1	Specifying Variability of Dynamic Software Product Lines using FODA Feature Models . . . . .	34
5.2.2	Specifying Runtime Contexts using Context-Aware Fea- ture Models . . . . .	36
5.2.3	Specifying Context-Aware Reconfiguration Behavior . .	37
<b>6</b>	<b>MAKI Transitions: Behavioral Aspects</b>	<b>38</b>
6.1	Introduction . . . . .	38
6.1.1	Utility-Based Description . . . . .	38
6.1.2	Goal Based . . . . .	39
6.1.3	Rule Based . . . . .	40
6.1.4	Rule Based plus Rule Learning . . . . .	40

6.2	Behavioral Modeling Based on ECA Rules and Description Language . . . . .	40
6.3	Behavioral Modeling Based on Min-Cost-Flow Optimization . .	42
6.3.1	Nodes . . . . .	42
6.3.2	Arcs . . . . .	43
6.3.3	Costs . . . . .	43
6.3.4	Flow Balance . . . . .	43
6.3.5	Bounds on the Flow . . . . .	44
6.3.6	Optimization . . . . .	44
6.3.7	Limitations and benefits . . . . .	45
6.4	Modeling Topologies under Transitions . . . . .	45
6.5	Modeling Transitions with Game Theory . . . . .	47
6.5.1	Importance of Decentralized Algorithms for MAKI Systems . . . . .	48
6.5.2	A Short Overview of Non-Cooperative Game Theory . .	49
6.5.3	Applying Game Theory to MAKI Systems . . . . .	50
<b>7</b>	<b>Conclusions</b>	<b>52</b>
<b>A</b>	<b>References</b>	<b>53</b>
<b>B</b>	<b>An OSI Model &amp; Terms Primer</b>	<b>59</b>

### Abstract

The Internet faces ever faster and stronger dynamics in the behavior patterns of its users and hence, in the imposed load and traffic. However, the various ‘mechanisms’ used within the Internet—communication protocols and their functional components, overlays, middleware, etc.—cannot be sufficiently adapted at runtime: parameter adaptation is common practice, but the replacement of a mechanism by one that is functionally similar yet more appropriate—for the benefit of performance and quality—is rare. Those rare cases are tediously engineered case by case since on-the-fly *transitions between similar mechanisms* are not promoted by today’s Internet construction principles.

In light of the considerations above, the present whitepaper advocates **mechanism transitions** as a fundamental new principle for the Internet and makes inroads into its modeling and specification.

The quest for a fundamentally more flexible Internet is not only fueled by trends regarding users and applications ‘above’, but also by innovations in the network technology ‘below’: Software-defined systems and networks emerge as an enabling technology for more flexibility, but cannot be sufficiently leveraged for more flexibility in the Internet as a whole.

The German Collaborative Research Center MAKI investigates appropriate models, concepts and methods as well as prerequisites and benefits in regard to mechanism transitions in the Future Internet. The present whitepaper comprises approaches to the modeling and specification of such mechanism transitions, both from a structural and a behavioral perspective, as developed over the last three years. The various other aspects of MAKI, such as investigations of particular sets of mechanism or monitoring and control aspects, are not covered here. The whitepaper starts by providing a more detailed discussion of the quest for Internet *mechanism transitions* and of related issues. It continues by introducing the basic MAKI architecture and terminology in comparison to those of the OSI standard and the Internet. In the sequel, different approaches to the general *structural and behavioral modeling and specification of mechanism transitions* are presented, as developed and used in MAKI. The presented research contributions were created with different purposes and focuses in mind; they represent important steps forward on our path towards a consolidated framework for mechanism transitions. In other words, they make inroads into a highly dynamic Future Internet that can cope with ever increasing dynamics.

We will use terminology from the ISO OSI standard as a starting point for introducing our terms and concepts. For readers with limited knowledge in this underlying standard, we provide a short OSI terminology primer in the appendix.

## 1 Mechanism Transitions: A New Paradigm

The Internet is known for its perpetual bandwidth famine: an increase in bandwidth provision is always followed by an increase in bandwidth demand—not to the least due to ‘hungry’ new application types (e.g., online games, P2P sharing, mobile video). In recent years, this well-known phenomenon was overlaid with a considerable increase in the *variety and dynamics* of these bandwidth demands. This latter effect is due to more heavily fluctuating *individual* usage patterns (a consequence of the growing peak demand of individual applications in response to the bandwidth famine), much aggravated by a correlation with *crowd hotspots* or *crowd use patterns*.

For better understanding, the following example illustrates this effect. A user may receive just a few sparse notifications on her smartphone while driving her car; seconds later, she may come to a full stop at a train crossing, turn to her device and trigger both a high resolution video stream and a file synchronization with the cloud in parallel. Firstly, such steep changes in bandwidth and other QoS demands have become common for end users of the general Internet and lately even of the mobile Internet. (Not long ago, bulky network traffic was the domain of confined and well-equipped enterprise networks.) Secondly, these fluctuations become much more dramatic if crowd hotspots emerge: in the example, many cars may have to stop and queue at the same train crossing, probably causing almost synchronous jumps in network demand. (Similar patterns may occur at other small or large crowd hotspots, such as commute hubs, city centers, sports venues, etc.). Thirdly, a trigger of a video stream or file download as in the above example may even concern the same source as an example of a crowd use pattern: a viral social media posting, live screening of a sports event, a security critical patch, etc. The example discussed in this paragraph serves as a short and simple illustration of the much increased Internet user dynamics that we face. Of course, high dynamics across the Internet as a whole also appear as higher fluctuations in the resources available to serve the fluctuating demands of a particular individual user or user crowd.

### 1.1 Resulting Future Internet Challenge

As the aforementioned example shows, the ‘load on the Internet’ may fluctuate by several orders of magnitude within fractions of a second. The structure and behavior of the network that has to master these dynamics, however, seems to be—relatively speaking—‘carved in stone’: Apps are compiled and linked with predetermined middleware components, leaving little room

for dynamic adaption, and lower layers are not built for on-the-fly reconfiguration either. Dynamic parameter adjustment within a given configuration is wide spread, but only two kinds of true dynamic reconfiguration—i.e., of mechanism transition—have gained a certain degree of common use over the last decades:

- Some heavy-bandwidth apps feature a built-in choice between several transport protocols. For instance in the context of video streaming, scalable video coding may seem to be a very adaptive approach as well, but it is largely a means for adapting the users' load, not the network behavior; at least, it 'makes the best' of a reduced load when forcing the user to accept it.
- The standardized network interface provides an abstraction layer between IP and the available MAC and physical layer connections (e.g., WiFi, G3/G4); the choice is usually made by the OS and often fixed at app startup time for the entire session; this is about to change along with the advent of G5 mobile networks, TCP handover, and multipath-TCP—and thereby the advent of dynamic handover among MAC layers, or concurrent use of several MAC layers, at runtime.

The latter of these developments is currently still the only one of wide-spread interest in which a runtime choice among several alternative options is made continuously, at runtime, in a way that allows all applications to benefit. As such, it is a unique sample realization of dynamic mechanism transition in the current Internet. This being said, it shows that there are no established principles nor general concepts and methods that support mechanism transitions. Moreover, it is a rather simple, dedicated, and specifically engineered case.

In summary of the above, one can state that (i) the dramatically increasing dynamics of the Internet leads to a quest for dynamic switching between protocols and other schemes, while (ii) only few, rather simple examples of corresponding approaches exist today, (iii) little knowledge exists about the relationship between 'conditions' (e.g., network load, signal strengths) and performance of individual protocols or communication schemes, (iv) cross-layer optimization between several dynamically adaptable layers is far from being common in the Internet.

This state of affairs is the point of departure for MAKI (Multi-Mechanism Adaptation for the Future Internet<sup>1</sup>), a Collaborative Research Center funded by the German National Science Foundation. MAKI aims at a future dynamic

---

<sup>1</sup>In German: 'Multi-Mechanismen-Adaption für das künftige Internet'

and adaptive Internet in which arbitrary components of the communication ‘stack’—all the way from app to ether or wire—can be switched at runtime to adapt to changing load and fluctuating network conditions. Obviously, such switching has to consider many boundary conditions, for instance:

- If a component is part of a distributed functionality (e. g., a network protocol or distributed functionality in a middleware), switching may have to be conducted synchronously at many nodes.
- The target component of the switching may also suggest (e. g., for performance reasons) or even require (e. g., for compatibility reasons) switching of other components.
- Switching may be too ‘expensive’ (e. g., in terms of temporary performance degradation, of required switching time) to be justified if a ‘current condition’ that suggests the switch cannot be expected to hold long enough for the switching to ‘pay off’ (even oscillation may occur).

The present whitepaper describes and discusses the general ‘mechanism transition’ approach that is key to all MAKI projects and introduces several approaches introduced for modeling and specifying concrete realizations of this approach. We will start by comparing the conventional OSI model for open systems interconnection with the current practice regarding the structure of computer networks and with the architecture advocated in MAKI as a basis for dynamic ‘mechanism transitions.’ Following this introduction, we will present the actual concepts for modeling and specifying ‘mechanism transitions’ and discuss their pros and cons. For the time being, these different approaches are concurrently applied and evaluated in MAKI in order to gain further experience.

## 2 From OSI via Internet to MAKI

### 2.1 The OSI Model as a Starting Point

The ISO standard *Open Systems Interconnection (OSI)* [61] consists of basic terminology, a seven-layer architecture, and a set of communication protocols and services. OSI is common knowledge among computer network specialists since more than three decades and, more importantly, it provides a sound basis for our work. Therefore, we retain the OSI terminology as far as applicable in this document. Given the broad variety of expertise and background of potential readers of this document, we provide a quick summary of the most

relevant terms and model elements of the general OSI model in Appendix B. We omit the OSI seven-layer model, which most computer network experts consider obsolete, and concentrate on the general model and terminology. Readers who are truly familiar with the terms communication protocol, communication service, protocol entity and protocol function, (N-)layer, service access point, request/confirmation and indication/response are *not* required to refer to Appendix B.

## 2.2 The Internet in Relation to OSI

The following developments in the Internet lead to deviations from the purist interpretation of the OSI model:

- **Fragmented ‘under-underlay’:** The overwhelming success of the Internet led to the integration of many network types; quite often, such network types exhibit specific peculiarities that require an elaborate network stack ‘below IP’, leading to more than two layers below the ‘classical layer three’ (network layer, i.e., IP); this led to a technology- and topology-driven fragmentation of the ‘under-underlay’ layers below IP that is today only partly reflected in the layer-three ‘underlays’ that follow the IP-specific structuring concepts (e.g., autonomous systems (ASes), IP-subnetworks)
- **Virtually separated networks:** The ‘hourglass’ shape of the landscape of Internet protocols (everything over IP – IP above everything, i.e., above every ‘sub’-network and above all lower-layer protocols) made IP ‘irreplaceable’ even though its design was suboptimal for newer application classes, in particular low-latency media streams of VoIP and IPTV; hence, virtually separate networks emerged where Telcos could manage this low-latency traffic via reservation schemes; the list of contributing technologies on different layers and of different impact leads from WDM and MPLS decades ago to SDN more recently.
- **Blurred layers, selective functionality:** As throughput requirements grew ever faster, cross-layer optimization considerations suggested a departure from the strict ‘distributed abstract machine’ concept introduced by OSI; many approaches advocated a pick-and-choose approach where applications would be supported by custom ‘stacks’ of selected protocol functions across all layers.
- **Balkanized middle layers:** As mentioned above, the inefficient and partly impractical OSI standards proposed for layers five and six strengthened



the success path of the 'Internet stack' where these layers are basically non-existent: applications are supposed either to connect directly to the transport layer (sockets) or to connect to application-layer protocols (e.g., HTTP, SMTP). Since the construction of more sophisticated distributed applications atop TCP is tedious and usually comprises a lot of recurrent functionality, there was a big demand for additional support in the 'free space' left over from the vaporized OSI-layers five and six. Four major kinds of support were (and are) provided in this respect:

- **Middleware:** Middleware realizes recurrent functionality that requires distributed realization on (more or less) all nodes involved in a distributed application; since most middleware is internally designed according to the 'blurring of layers' paradigm (see above) yet clearly separated from applications and underlying (Internet) transport protocols, it can be viewed as a 'single fat layer'; in light of resource-constrained special-purpose or mobile nodes, recent approaches targeted lean approaches by following the pick-and-choose paradigm.
- **Middleware services:** Recurring functionality that can be provided on selected nodes for a whole network domain (such as time services, authentication services) are often realized as middleware services; service provision to application entities usually follows a client-server communication scheme.
- **Overlays:** The term 'overlay' is commonly used instead of 'middleware' if the recurring functionality is provided through clever topology construction and clever repartition of that functionality across the (distributed) entities providing it; overlays are typically designed with scalability as a major design consideration. Peer-to-peer file sharing systems can be considered to mark the starting point of overlay concepts.
- **Communication paradigm driven solutions:** The advent of novel communication paradigms (e.g., remote procedure call, application level multicast (ALM), publish/subscribe (pub/sub)) along with competing realizations (e.g., various overlay routing approaches), has been a major driver for novel middleware (providing the functionality in a 'communication protocol' oriented way) and overlays (providing clever routing approaches and topologies); this driving role of communication paradigms merits special mention as one of the forces that pushed the 'Internet reality' of today further away from the 'perfect OSI world' of the 1990s.

- **Quest for parallelization:** Finally, the protocol / layer structure is further challenged by the increasing parallelization that results from physical limits of ‘Moore’s law in the classical sense’. CPUs, multi- and many-core architectures as well as ever more powerful network interface cards (NICs) hold the promise to help counter the ever increasing throughput demands of network stacks. At the same time, they re-introduce the need for clear separation (e. g., at the boundaries to GPUs or NICs).

## 2.3 Towards the MAKI Internet Model

As a novel approach that aims at a paradigm shift of the Internet, the MAKI Internet model has to reflect (i) the past, i. e., the OSI model and its tailoring to the Internet as it occurred some three decades ago, (ii) the present, i. e., the realities of the present Internet as summarized in the last section, and (iii) the necessities imposed by the key concepts ‘mechanism’ and ‘transition’ and the considerations they impose.

Obviously, the encompassing notion of ‘mechanism’ must be refined such as to cover not only protocols and services as in the OSI case but the full variety of functionality realized by cooperating components in today’s Internet. In addition, this concept of abstract mechanisms and their incarnation in cooperating runtime instances must both be enriched by all the necessary elements, both structural and behavioral, that are required in order to enable mechanism transitions. Transitions, in turn, must be conceived such that they can be specified and realized by parties (i. e., vendors and communities) that are independent from mechanism providers—and they must be extensible as new candidate mechanisms for a transition are added.

# 3 Modeling MAKI Mechanism Transitions

## 3.1 Requirements Imposed by the Reality of Today’s Internet

The previous chapter argued how and why the current reality of Internet-based networks deviates considerably from the purist OSI general model and from the conventional four-plus-one-layers Internet model. We will condense the arguments brought forth in Section 2 into the following four characteristics that describe how today’s Internet deviates from clean models that supposed to be its foundation, since any model of the Future Internet should be able to reflect the reality of the Internet.

- (I1) **Fragmented network:** Different under-underlays require different protocol stacks below IP; vertically sliced ‘virtual networks’ are required for proper handling of different traffic classes, in particular in response to VoIP and IPTV, with SDN as the latest movement; these fragmentations complement the known AS/subnet partitioning
- (I2) **Pulverized protocols:** Mainly for performance reasons, the strict hierarchical layering concept has been increasingly traded in for pick-and-choose models of protocol (function) selection across layers.
- (I3) **Clustered functionality:** While trend (I2) led to highly modular, i. e., fine-grained self-contained functional blocks, other developments had a reverse effect, i. e., led to a clustering of functionality with sharp external boundaries but blurred internal structure; e. g., self-contained middleware and overlays encapsulate rather complex intertwined sets of functions, isolating these parts from the rest of the network stack; powerful computer-like NICs and GPUs and multi-/many-core trends, all available on a single computer node, suggest the network stack to be partitioned among these ‘processing elements’.
- (I4) **Complex topology:** The various overlay approaches (e. g., for server-less high-availability storage, for collective video transport) and application-layer routing concepts (needed in support of, e. g., pub/sub and ALM) mark a trend away from the classical point-to-point design and specification of communication protocols, where the main considerations concerned two interacting parties in the system.

### 3.2 Requirements Imposed by Mechanism Transitions

As discussed in the introduction, MAKI aims at a substantially more flexible, dynamically adaptable Internet. The term Internet is thereby conceived as a superset of all interconnected communication systems<sup>2</sup> worldwide. The dynamic adaptation is meant to extend substantially beyond parameter adaptation: switching from one ‘component’ to another one at runtime shall be specifically supported. The (often distributed) functionality represented by a switchable, i. e., exchangeable, component is denoted as **mechanism** hereafter; as laid out in the previous chapters, the concepts *mechanism* and *transition* (see below) represent the fundamental terms and concepts conveyed in this whitepaper. More precisely speaking, a mechanism is realized in the

---

<sup>2</sup>The term communication system is used to denote a computer network together with its distributed computation support (e. g., overlays, middleware).

form of a set of distributed runtime code instances (non-distributed scenarios, i.e., a set of local instances or even a single instance, are comprised in this definition as special cases). The cooperating instances actually *provide* the specified mechanism functionality in the Internet. In accordance with OSI terminology, we denote these runtime code instances as (*mechanism*) *entities*. A mechanism—more precisely: its entities—may be dynamically replaced by a different set of distributed entities that realize a different, functionally similar mechanism. (For details see further below.) The act of switching (exchanging one mechanism, i.e., the set of mechanism entities by another one) is denoted as **transition**.

In comparison to the occasional approaches to dynamic switching that exist in today's Internet, the *mechanism transition* approach pursued in MAKI is thought of as a fundamental principle for the Future Internet, permitting much wider spread, easier and more consistent development, and most importantly interoperability in open systems. Towards these aims, the MAKI concept also exhibits subtle differences from occasional existing approaches. This is exemplified by a look at prominent examples of existing approaches: those that permit switching between UDP and TCP, between TCP and RTP/RTSP, and between different data link interfaces such as WiFi, 3G, 4G, 5G, and wired connections. In these approaches, it is the party using a mechanism or the local operating system that decides about switching: for instance, apps may decide about switching between different transport layer mechanisms, the operating system may decide to switch between data link layer mechanisms, and in the case of multipath-TCP, it is the higher layer (TCP) that decides about which lower layer protocols to use. (OSI literates may note that functionality like that of multipath-TCP was by and large anticipated in the OSI model under the term 'downward multiplexing', except that dynamic load balancing had not been explicitly addressed by then.) In contrast to these known approaches, the concept of MAKI mechanism transition assumes a dedicated control logic to surround the (source and target) mechanisms involved in a transition—notwithstanding the possibility for a third party (e.g., app, higher layer, OS) to influence the control logic, i.e., to cause a transition indirectly (see 'utility functions' further below).

Before introducing more details of the MAKI reference model, the reasoning that governs the MAKI model must be considered in more detail. Consequences of the principles guiding MAKI for the correspondingly revised Internet model will be enumerated as (M1) through (M5) below.

**(M1) Representation of static aspects of the dynamic control logic for transitions, its modularization, and federation:** The necessary computational

logic as discussed above must be explicitly represented as elements of a MAKI-compatible architectural model; a modularization of this logic should be supported and reflected: MAKI suggests the well-known M-A-P-E cycle for separation of concerns here (see below); on the other hand, a federation of conceptual components across layers, and of the corresponding entities (run-time instances, see above) across layers and network nodes must be reflected in the model.

In open networked systems (following the ‘open systems’ vision advocated by both the OSI and Internet communities), the interdependencies and interrelations between transitions and their surroundings cannot be completely foreseen at the time a mechanism is specified and implemented; they may not even be fully known at the time a transition between source and target mechanism is specified and implemented. Examples for such dependencies that cannot be (fully) foreseen at mechanism or transition design time include, in particular: (i) dependencies to/from other mechanisms and transitions, both ‘hard’ (e.g., mutual exclusion due to incompatibility) and ‘soft’ (e.g., mutual effects on the performance of the other mechanism); (ii) requirements imposed by parties using (sets of) mechanisms, i.e., apps and ‘higher’ mechanisms. This leads to the quest for a human- and machine-readable specification language for the transition-governing M-A-P-E cycle. (We will use the term ‘abstract specification’ to refer to the hybrid, i.e., easily human and machine readable quality of such a language.) This quest can be further detailed as follows.

**(M2) Abstract specification of dynamic aspects of the transition control logic:**

The transition control logic represents the behavior of a particular transition between mechanisms; it concerns the necessary preparation, state and function transition, and re-instantiation steps and may provide for simultaneous operation of both mechanisms during parts of the transition. This logic should ideally be available as abstract specification. Since it may concern only a particular transition and does not have further external effects apart from the transition itself, there is a temptation to avoid formal specification and modeling. However, the use of formal approaches provides a good interface where human and machine based reasoning meet; in addition, it is known to improve and simplify maintenance and to enable generalization (here: to other transition control logic).

**(M3) Abstract specification of interdependencies:** Regarding point (i) in (M1), reasoning about mutual dependencies between transitions and about ‘optimal’ cross-transition configurations should also be possible using an abstract specification.

**(M4) Abstract specification of the transition decision logic:** Probably the biggest challenge in the new paradigm proposed by MAKI concerns the enabling of decisions about transitions: when would which transition be triggered and performed. While simple rules suffice for simple cases, the mid- and long-term goal must be to provide a framework for a sophisticated control loop. This control loop should ideally take into account the demand side, i. e., current and expected future load situation and user desires (one may call it the problem space), and the supply side, i. e., dynamic adaptation possibilities of the Future Internet (one may call it the solution space).

In the research work of the MAKI team and based on related work, the above-mentioned control loop was thoroughly investigated, guided by the classical measure-analyze-process-execute (M-A-P-E) cycle introduced in the autonomous computing context by IBM in 2004. In this context, the following major aspects turned out to be suitable for capturing the design and solution spaces:

- *Environment conditions*, i. e., parameters that describe the load situation of several alternative mechanisms
- *Mechanism performance characteristics*, i. e., quality-of-service parameters describing an aspect of the solution space suitable for comparing the performance of alternative mechanisms; note that mechanism performance may be inversely described by means of the *cost*, i. e., overhead (resource demand or similar), associated with providing a desired quality-of-service; mechanism cost can equally be used for comparison of alternatives in the solutions space
- *Transition costs* capturing the effort required for executing a transition (e. g., resource demands, temporary performance degradation); obviously, transition costs must be traded off against expected performance gains
- *Utility expressions* providing a crucial dual-purpose ‘link’ between problem space (environment conditions) and solution space (performance characteristics / costs): since different mechanisms will usually cater for different performance characteristics to a different degree, and since the improvement of one quality-of-service parameter may degrade another one, utility expressions must be used in order to determine the contribution (weight) of each performance parameter to the overall performance. Moreover, different users may have different preferences and different importance, i. e., weight, as mechanisms cater for several users. In this respect, utility expressions may be used to integrate these

(maybe conflicting) user interests into a single ‘formula’ that maps the complex problem space into what is to be optimized in the solution space

- *Predictors*, i. e., means for capturing time as additional and independent dimensions of the mechanism transition dynamics; decisions that are solely based on current demand or load may turn out to be inappropriate by the time the transition is carried out; in other words: without considering (and hence, predicting) future situations, mechanism transitions may be suboptimal or counter-productive, or even lead to oscillating behavior; while predictors for expected future situations (future environment characteristics and utilities) are difficult to achieve and may still be inexact, they can considerably improve system performance

**(M5) synchronization of dependent transitions:** Two major kinds of dependencies between transitions must be carefully considered when effectuating a transition, leading to a (potential) need for synchronization: (i) intra-transition (cross-network) synchronization concerns the fact that mechanisms are provided by a distributed set of (often identical) components (cf. protocol entities in the conventional OSI model, Appendix B); they may lead to the need for effectuating a particular intra-transition protocol among the entities effectuating the transitions on different nodes; (ii) inter-transition (cross-node) synchronization concerns interdependencies as discussed under (ii) in (M1); they may lead to the need to carry out several transitions on a single node in parallel or even in lock-step.

In the next chapters, we will first establish a normative MAKI reference model, mainly intended for establishing concepts and terminology. The subsequent chapter will then introduce particular modeling and specification techniques that were developed or furthered in the context of MAKI research. Their basic reason of existence is for simplifying a developer’s task to realize MAKI conformant mechanisms, multi-mechanisms (for the term see below), and control cycles that drive transitions.

## 4 The MAKI Reference Model

### 4.1 The Structural Model

The first consolidated version of the *structural* MAKI Internet reference model is introduced below by means of annotated architectural figures. The requirements (I1) through (I4) resulting from today’s Internet (see Section 3)

are mainly reflected as follows. First of all, the strict architecture put forward by OSI (see Appendix B) with its rigid separation in layers and its singular granularity of services plus corresponding protocols is replaced by a more universal modular architecture where the concept of a *mechanism* is the design center. Mechanisms can be freely composed into distributed caller-callee and composite-component structures, and such compositions may be encapsulated into (OSI-like) distributed virtual machines and structures that may serve as self-contained layers if desired. At the same time, node and network structures can be reflected in the model by means of node containers called processing and storage elements, and by means of network containers called subnetworks which can be arbitrarily overlapping. Finally, (I4) is reflected by explicitly introducing *topologies* as first-order objects in the mechanism related model.

The MAKI-specific requirements (M1) through (M5) discussed in Section 3 are reflected as follows. The concepts mechanism, (mechanism) entity, and transition as sketched in the beginning of Section 3.2 are introduced systematically as part of the MAKI reference model. (M1) through (M5) inform details of these concepts and, in particular, further structural elements of the MAKI reference model as introduced below. Behavioural aspects are reflected by means of ‘placeholders’ that mark up the relationship between structural and behavioral elements.

### 4.1.1 Basic Reference Model for Mechanisms

The relationship between mechanisms, (mechanism) entities, and topologies (cf. section 3.2) is depicted in Figure 1.

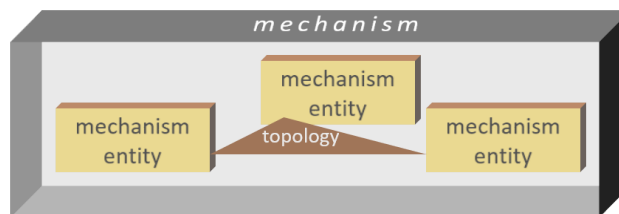


Figure 1: Mechanism (conceptual level) with corresponding (runtime) entities and topology (architectural level)

For a more concise description, we introduce a set of informal definitions as follows.



**Definition 1: MECHANISM (Key Concept)**

A *mechanism* is a confined conceptual element of a networked system that is bound to a realization as cooperating functional units; the special case of an isolated realization (single-node) is permitted.

Examples: services in the OSI sense (cf. Appendix B) represent a special case of mechanisms; other examples include protocol functions in the strong and in the relaxed OSI sense (cf. Appendix B), conceptually confined functionality of a middleware or of an overlay (e.g., those that realize a communication paradigm), comprised or auxiliary functionality in support of these, and the entire middleware or overlay as such.

**Definition 2: (Mechanism) Entity**

A *mechanism entity* is an element of a communication system that realizes node-specific functionality at runtime that pertains to a mechanism; the entity and cooperating peers together realize the distributed cooperative provision of a mechanism.

In analogy to the OSI terminology, we consider mechanisms as abstract concepts, the functionality of which is often realized by cooperating runtime modules; these runtime modules are denoted as (mechanism) entities, again in accordance with OSI. In analogy to the OSI terminology, the term mechanism is often commonly used for the sake of simplicity in cases where a distinction between mechanism and mechanism entity would be more precise.

**Definition 3: (Mechanism) Topology**

A *mechanism topology* is the meshwork of (logical) connections between mechanism entities. The default representation is a graph in which the vertices represent the entities that realize a mechanism.

We already pointed out that in today's Internet, complex overlays and other multi-party structures complement the former 'point-to-point-centric' conceptualization of (OSI-model-like) services and protocols. In light of the much increased relevance of the multi-party structures pertaining to the realization of mechanisms, *topology* was 'promoted to first-class citizen' in the MAKI model.

### Definition 4: Communication Mechanism

A mechanism is refined as *communication mechanism* if it is involved in the handling or forwarding of user data.

### Definition 5: Control Mechanism

A mechanism is refined as *control mechanism* if it is involved in the operation of other mechanisms in the sense of auxiliary functionality.

In the OSI reference model, protocols and services were supposed to serve the purpose of forwarding packets between layers and across nodes. A look at, e. g., the IP protocol and its companion protocols (a. k. a. IP auxiliary protocols) such as IGMP and ICMP shows the importance of another kind of protocols and services: those that are *not* directly involved in the handling and forwarding of data ‘packets’ (PCI plus PDU in OSI terms). Therefore, the distinction of two classes of mechanisms (communication and control) is introduced. Important examples for control mechanisms are *monitoring mechanisms* and *topology control mechanisms*. Note that the attributes ‘communication’ and ‘control’ may be used as prefix for other concepts and constituents of the MAKI reference architecture, such as mechanism entity and multimechanism.

### 4.1.2 Possible Relationships of Coexisting Mechanisms

According to the definition of a mechanism, a networked system contains a large number of mechanisms (although today, the vast majority of them is not perceived, let alone specified and realized, as a mechanism in the sense of the MAKI reference model. The large number of mechanisms found already in today’s communication systems can be inferred by just looking at the functional building blocks of communication protocols that are called protocol functions in OSI terms: rate, flow, and congestion control, addressing, acknowledgment, timeout, checksum calculation, upward and downward (de-)multiplexing, and so on. Middleware, overlay networks, etc. are not even considered in this exemplary list. The vast amount of mechanisms today will obviously grow further in the future. Therefore, it is useful to introduce major categories of relationship among coexisting mechanisms, worthwhile to discern. Such categories are introduced below.

**Definition 6: Super-Mechanism / Sub-Mechanism**

If an ‘encompassing’ mechanism applies an ‘encompassed’ mechanism as part of its specification, the former is concretized as *super-mechanism* and the latter as *sub-mechanism*

For communication mechanisms, *encompassing* means that processing of data structures as exchanged in the network by the super-mechanism’s entities continues after the usage of the sub-mechanism is terminated.

Note that a sub-mechanism may act as a super-mechanism in a different relationship and vice versa. The analog ‘transitivity’ applies to the following definition.

**Definition 7: User-Mechanism / Provider-Mechanism**

If a ‘higher’ mechanism applies a ‘lower’ mechanism as part of its specification, the former is concretized as *user-mechanism* and the latter as *provider-mechanism*. In contrast to the relationship between super- and sub-mechanisms, ‘higher’ means that the distributed handling of data structures in the network (transportation, storage-and-retrieval) is provided by the ‘lower’ mechanism in lieu of the higher mechanism.

The last two definitions are illustrated in Figure 2 below. Software engineers will note that they reflect wide-spread relations common to, e.g., object-oriented programming.



Figure 2: Mechanisms in super-sub and user-provider relation

**Definition 8: Functionally Equivalent Mechanism**

A mechanism  $M_2$  is called *functionally equivalent* to a mechanism  $M_1$  if any user- or super-mechanism conceptualized for the use of  $M_1$  does not have to be modified if  $M_1$  is replaced by  $M_2$  and if under  $M_2$ , the user- or super-mechanism retains its functionality

Obviously, equivalent mechanisms have to offer exactly the same interface to their user- or super-mechanisms, where ‘same’ refers to both the syntax (signature) and behavioral semantics at the interface, so that only non-functional

differences such as different performance functions are exhibited ‘upward’ in the communication system. Functional equivalence is very rare in today’s Internet and cannot be expected to be common in the near future. Therefore, another term is introduced for the functional comparison of mechanisms.

##### Definition 9: Functionally Similar Mechanism

A mechanism  $M_2$  is called *functionally similar* to a mechanism  $M_1$  if core functionality—defined as the majority, i.e., more than 50% of the overall functionality—of  $M_1$  and  $M_2$  can be formally or informally defined in identical terms and if additional functionality can be specified that provides functional equivalence according to Definition 8.

In cases where communication services are exchanged already today (see, e.g., WiFi vs. LTE vs. Ethernet or TCP vs. UDP vs. RTP), functional equivalence is not 100% present. Rather, the functional differences must be shielded or mitigated (e.g., by the virtual network interface card, VNIC). The network access layer below IP is an example for shielding, and multi-protocol video streaming solutions provide mitigation. Still, shielding and mitigation cannot be provided if mechanisms are too different, which brings about the term ‘functional similarity’ that is crucial to the definition of transitions in the next section.

Since multimechanisms shall be in a way that includes the potential shielding of functional differences between mechanisms, the definition of functional similarity above is crucial to the MAKI reference model.

##### Definition 10: Dependent / Independent Mechanism

Coexisting mechanisms are called *dependent* if they exhibit relationships of type super-/sub-mechanism or of type user-/provider-mechanism. Coexisting mechanisms are called *independent* if they are neither dependent nor functionality equivalent nor functionally similar.

**Source and target mechanism.** In the context of transitions, the terms source mechanism and target mechanism will be applied to denote the mechanism from which and to which a transition is carried out, respectively. Since these terms depend on the context and do not classify mechanisms as such, they are not included in the list of numbered definitions provided in this whitepaper, which constitute the core of the MAKI reference model.

## 4.2 Behavioral Model: Transitions

In light of the argumentation and definitions provided so far, we can immediately proceed to the definition of the second central concept of the MAKI reference architecture:

### Definition 11: (Mechanism) TRANSITION (Key Concept)

A (*mechanism*) *transition* is the functional replacement of a (source) mechanism by a functionally similar or equivalent other (target) mechanism in a running communication system, without causing an error condition in any dependent mechanism.

A transition may concern one, several, or all super- or user-mechanisms of a communication system for which the source mechanism acts as sub- or provider-mechanism, and for which it is replaced.

In the particular case where the source mechanism is the only super- or user-mechanism of a dependent sub- or provider mechanism, such a dependent mechanisms may be halted or terminated if the target mechanism of the transition will replace or restart it.

The introduction of the concept of transitions raises the issue of agents that trigger or perform transitions and of the corresponding decision making. Section 4.3 and subsequent chapters will emphasize these behavioral aspects. For now, we introduce one definition in order to be able to capture this issue in the subsequent discussions.

### Definition 12: Transition Control Logic

The *transition control logic* is the decision making and management functionality required for deciding about transitions and for enabling and steering their execution.

Transition control logic consists of a partial or complete cycle through the steps of the M-A-P-E cycle: monitoring (collection of evidence), analysis (computing of system status and transition cost), planning (assessment of the status after transition, core decision making), and execution (provision for and steering of the actual transition).

Considering the vision of a future dynamic Internet with frequent mechanism transitions, the transition time and ‘smoothness’ will be a major aspect. These considerations suggest distinctions between transitions as follows.

**Definition 13: Sharp Transition**

In a *sharp* transition, the source mechanism is terminated before the operation of the target mechanism starts.

As specified in the transition definition, the termination and start of a mechanism concerns the role of the source and target mechanisms for a super- or user-mechanism; it does not necessarily imply the termination or start of mechanism entities in terms of executable code. This applies to the next two definitions, too.

**Definition 14: Handover Transition**

In a *handover transition*, the source mechanism is terminated after the target mechanism has started and if the overlap period serves the purpose of reduced performance slump

**Definition 15: Coexistence Transition**

In a *coexistence transition*, two or more functionally equivalent or similar mechanisms are permanently in use for the same super- or user-mechanisms and if the transition control logic determines the distribution of tasks among these mechanisms as opposed to triggering the (sharp or handover) transitions as such.

**Additional structural component.** The above definitions of transitions and related concepts provide the basis for introducing an additional structural component that makes the possible differences between multiple transitions and the transitions themselves transparent to super- and user-mechanisms.

**Definition 16: Multimechanism**

A *multimechanism* encapsulates several mechanisms and the transitions between them; it comprises

- two or more functionally similar or equivalent mechanisms;
- a single mechanism interface towards super- and user-mechanisms (the multimechanism appears as a single mechanism to those mechanisms) and, in case of functionally similar mechanisms, the necessary code for ‘equalizing’ the functional differences between comprised mechanisms to the necessary extent for ensuring the properties of a transition as specified;
- an interface towards external control logic for supplying information about internal states and measurement and for receiving transition decisions or decision aids;
- internal transition control with mandatory support for execution of transitions and optional measurement, analysis and planning parts.

The concepts of sub-/super-mechanisms and user-/provider-mechanisms may be lifted to multimechanisms, resulting in sub-/super-multimechanisms and user-/provider-multimechanisms, respectively.

Similar to today’s Internet where several services and protocols may expose cross-layer dependencies, particular combinations of dependent mechanisms may be favorable while others may be unfavorable or even impossible due to incompatibilities. Such dependencies cannot be hidden by means of the multimechanism concept; they must be considered by control logic that is external to the multimechanisms and considers these dependencies. This is a major reasons why multimechanism-internal transition control logic is optional except for the part that *executes* transitions.

This has two reasons: 1. the instantiation of a multimechanisms in the form of distributed entities may require different kinds of (or no) control logic to be realized in different entities, 2. overarching (‘cross-layer’) control logic may be used for several multimechanisms. For the same reason, a definition for a component such as *topology control entity* is not provided here.

### 4.3 Behavioral Model: Transition Decision

The quest for transition control logic as defined above was mainly put forth in requirement (M4); it was reflected in the definitions the corresponding notion and of multimechanisms above. A crucial part of control logic, and thereby of the M-A-P-E cycle, are the *decision making processes* that trigger transitions.

Often, the decision concerns a set of transitions for several mechanism components on several network nodes, in which case it may be necessary to decide which entities shall be in the set; finally, the decision may concern a set of transitions for different but interdependent mechanisms – which will again in turn be realized as a set of interconnected components.

**Holistic Model and Function for Decision Making:** In the remainder of this section, we will introduce the parts of the MAKI reference model that provide the general nature of a transition decision function, denoted as *utility function*. As usual for a reference model, the corresponding definitions will attempt to capture—to the best of the experience and knowledge of the authors—all aspects, characteristics, and parameters that can be taken into account in a transition decision. Therefore, the decision function will be composed of several auxiliary functions.

As the remaining chapters of this whitepaper illustrate, actual models applied for specific purposes in the MAKI context concentrate on more specific aspects and are hence driven by much more focused decision making concepts. Thereby, auxiliary functions are often reduced to single values or simple terms in the utility function. Such simplifications are also suggested since present decision making techniques (introduced in subsequent chapters) require simpler utility functions.

As to the encompassing utility function, auxiliary functions, and further inputs to the necessary terms and definitions for the ‘holistic’ model and function introduced below, we can refer to the context of (M4), where most of the key functions and input were already named as part of the problem description: **environment characteristic** (subsuming everything that describes the load and ‘surrounding’ situation of a mechanism) (mechanism) **performance characteristic** (quality-of-service parameter), **mechanism cost** (inversely characterizing a mechanism in terms of associated effort, e.g., resource consumption), **transition cost**, and **utility** (linking all inputs and functions together in weighted expressions where weights reflect user requirements and priorities). We will now provide more concise formal specifications as part of the MAKI reference model.



**Definition 17: Environment Characteristic**

An *environment characteristic* of a mechanism is a variable  $E_i$  or vector of variables  $\vec{E}_i$  that influences the performance of a mechanism yet is not under direct control of that mechanism; the variable or each vector component takes on a numeric, binary, or enumeratic (class) value at any given time of the operation of the mechanism.

**Definition 18: Performance Function  $\mathfrak{P}\mathfrak{F}$** 

The quality of service provided by a mechanism is modeled by means of a set of *performance functions*  $\{\mathfrak{P}\mathfrak{F}_i\}$ , where each performance function  $\mathfrak{P}\mathfrak{F}_i$  maps a set of environment characteristics  $\{E_j\}$  onto a particular quality-of-service variable  $Q_i$  or cost variable  $C_i$  of the mechanism; in the first case, the performance function describes an aspect of how well the mechanism renders its service, in the second case, it describes the amount of a relevant resource or effort required for service provision under the given set of environment conditions.

While the reference model is meant as an abstract description, one can easily infer typical resources for obtaining the above-mentioned data. Environment characteristics must usually be acquired by means of monitoring (measurement); if they are related to mechanisms ‘in the environment’ of the multi-mechanism(s) for which transition decisions are sought, then they may be obtained from these mechanisms upon request.

Performance characteristics of an *active* mechanism may either be monitored or derived from the performance functions based on measured (or obtained, see above) environment characteristics. Performance characteristics of an *inactive* mechanism (usually: a candidate mechanism of a considered transition) can only be obtained via performance functions, executed with measured (or obtained) environment characteristics as input. Working out a performance functions is a considerable challenge, usually subject to intensive research. In some cases, they can be derived from research publications.

**Definition 19: Transition Cost (Aggregation) Function  $\mathcal{C}\mathfrak{F}$**

A *transition cost function*  $\mathcal{C}\mathfrak{F}$  maps boundary conditions of a transition under consideration (such as the number of components employed or available resources) onto a numeric, Boolean, or enumerated value suitable for trading off transition gain—in terms of utility gain—against transition cost. Different categories of transition penalties may have to be considered, requiring different cost functions and a transition cost aggregation function. For decision making, the relevant cost (aggregation) function must yield the same output value type as the relevant utility function.

**Definition 20: Prediction Function (predictor)  $\mathfrak{F}\mathfrak{F}$**

A *prediction function*  $\mathfrak{F}\mathfrak{F}$  (predictor for short) can be defined pairwise for a performance function  $\mathfrak{P}\mathfrak{F}$ , taking the same parameters plus an additional one denoted as *time offset*; this time offset determines the time in the future for which a predicted performance for a given mechanism shall be computed. A prediction function may take on the form of a probabilistic estimator and yield additional statistical information such as probability distribution function, variance, etc.

It should be noted that lacking persistence of predicted future environmental conditions as well as oscillations may hamper or foil transitions. The gain drawn from a transition depends heavily on the future development of the environmental conditions; conditions that favor a target mechanism at a given point in time may persist or become even more favorable, and hence favor that mechanism further; in such a case, a transition may even pay off in case of high transition costs. If, however, the environmental conditions evolve quickly in a direction that favors a different mechanism, even low-cost transitions may be counter-productive.

The simplest conceivable predictor is one that assumes the current environmental conditions to last (independent of the time offset). On the other side, sophisticated machine-learning based predictors can be imagined which can be used for effectuating a transition in anticipation of an upcoming set of environmental conditions

**Definition 21: Utility (Aggregation) Function  $\mathcal{U}\mathfrak{F}$** 

A *utility function*  $\mathcal{U}\mathfrak{F}$  maps different weighted performance functions onto a single output value  $U$ :

$$U = \mathcal{U}\mathfrak{F}(\mathfrak{P}\mathfrak{F}_1(e_{11}, \dots, e_{1m}) * w_1, \dots, \mathfrak{P}\mathfrak{F}_n(e_{n1}, \dots, e_{nm}) * w_n);$$

different user- or super-mechanisms affected from the same transition - and for top-layer mechanisms: different users may provide different utility functions; in this case, a utility aggregation function must combine different weighted utility functions into a single value.

As discussed under (M4), a transition decision based on nothing but performance and cost functions is likely to be impossible or arbitrary: sets of performance functions and cost functions may yield contradicting values for different quality-of-service variables (e.g., for a distributed data storage case, a transition may promise to improve access time but reduce availability); therefore, a tradeoff between conflicting aspects must be achieved, which is the purpose of the utility function. For the analogous reason, an aggregated cost function is necessary if several cost functions exist.

Even in the case of two different transitions that both improve over a status quo, but with an emphasis on different performance characteristics, the transition decision cannot be taken without comparing the utility of these performance characteristics.

It is necessary to reason about the cumulative utility which a mechanism provides to a set of users (apps or ‘higher’ mechanisms); to this end, users must provide individual utility functions that map the set of considered quality-of-service variables to a single utility variable. For the integration of all utility functions of all users currently using a mechanism, the utility functions must be normalized and combined; normalization can be achieved by specifying constraints on the utility functions (such as an integral maximum), combination can be specified in a dedicated utility aggregation function (that may consider different weights or priorities for different user classes).

**Actual decision making.** Provided that comparable utility and cost functions for source and target mechanisms yield the same types of values and that the functions are designed such as to make these values truly comparable, the condition for a transition cannot be formulated quite simply: Let  $U_S$  and  $U_T$  denote the utility of a source and a target mechanism, respectively, obtained as output of the utility (aggregation) function; let further  $C_{S \rightarrow T}$  denote the transition cost for the transition from the source to the target mechanism as considered; then a transition is (in principle, see below) favorable if the target mechanism utility is larger than the source mechanism utility plus the

---

transition cost, i.e. if  $U_T - C_{S \rightarrow T} - U_S > 0$ . If several transitions are possible, the one with the largest value of the above subtraction is to be favored.

Obviously, the decision making as above does not yet properly take into account time: a favorable utility of a target mechanism weighs more if the performance functions can be expected to remain favorable for a longer time. If this aspect is to be reflected in the weights of the utility function then the weights must be considered as appropriate functions instead of single values. Again, obtaining these functions—and assessing the expected development of utility over time—is difficult issue and requires targeted research.

**Outlook on remaining chapters.** The present chapter introduced the MAKI reference model for communication systems based on the core concepts of *mechanisms* and *transitions*, providing a vision for a highly dynamic Future Internet. Both structural and behavioral aspects were addressed.

As common for reference models, the striving for comprehensiveness led to quite a number of concepts in the structural model and to a behavioral model that is quite far-fetched with respect to the knowledge embedded in the performance, cost, and utility functions defined. Therefore, we will complement the reference architecture with findings from a range of subprojects within the MAKI collaborative project. Thereby, Section 5 addresses issues of structural modeling, and Section 6 presents a number of approaches to the behavioral aspects of mechanism transitions.

## 5 MAKI Transitions: Structural Aspects

This chapter resumes structural issues of mechanism centered, transition enabled highly dynamic communication systems.

Section 5.1 presents a metamodel-based approach to structural modeling, making considerable inroads into the realization of the structural part of the MAKI reference model.

Section 5.2 emphasizes an important particular aspect, namely the complexity of composite structures consisting of super- and sub-mechanisms, user- and provide-mechanisms, and functionally equivalent or similar mechanisms comprised in multimechanisms. Considering possible combinations due to (at first sight) independent transitions in these potentially quite complex composites, unfavorable and functionally incompatible combinations will emerge. In the light of this particular problem, Section 5.2 provides an approach based on so called feature models.

## 5.1 Structural Modeling Based on Metamodels

In this section, we introduce metamodeling as a technique for describing the structure of MAKI-compatible systems. We illustrate this technique using three different sample system types.

### 5.1.1 Definitions

The following paragraphs introduce basic concepts related to meta-modeling [45]. A *metamodel*<sup>3</sup> describes the essential parts of a system under consideration. It consists of *classes*, which may have typed *attributes* (e.g., real-valued, integer-valued, or enumeration type-valued) and *methods*. An *abstract class* (depicted using italicized font) cannot be instantiated. To specify relations between classes, two connection types are employed: inheritance links and associations. *Inheritance links* are denoted by triangle-headed links and specify *is-a* relationships between classes. *Associations* are denoted by arrow-headed lines and specify *has-a* relationships between classes. Each association comprises two *association ends*, each possessing a descriptive *role name* and a *multiplicity*, which specifies how often a particular association may be instantiated at runtime. *Composition* is a special type of association, which specifies that one class, the containee, has a part-of relationship to another class, the container. The container end of such an association is depicted as solid black diamond.

In the following, we present these modeling elements using three examples of MAKI-compatible systems, originating from different areas of the communication system domain. As metamodeling framework, we employ *Ecore* [52] which is part of the *Eclipse Modeling Framework (EMF)*. One of the benefits of *Ecore* is that widely used mappings to object-oriented languages such as Java already exist.

### 5.1.2 Example 1: Wireless Sensor Networks

A *Wireless Sensor Network (WSN)* typically consists of a large number (hundreds to thousands) of small, cheap, and battery-powered sensor nodes, which are for example used to collect environmental data or to help during disaster recovery [46]. A WSN may be partitioned into several regions, exposing different characteristics (e.g., moving vs. static obstacles).

Figure 3 depicts a metamodel for the WSN application domain. A *WirelessSensorNetwork* consists of (sensor) *Nodes* and *Links* between them, which are grouped into (network) *Regions*. A *Node* has the real-valued attributes

---

<sup>3</sup>The following definitions conform to the Meta Object Facility (MOF) standard [24].

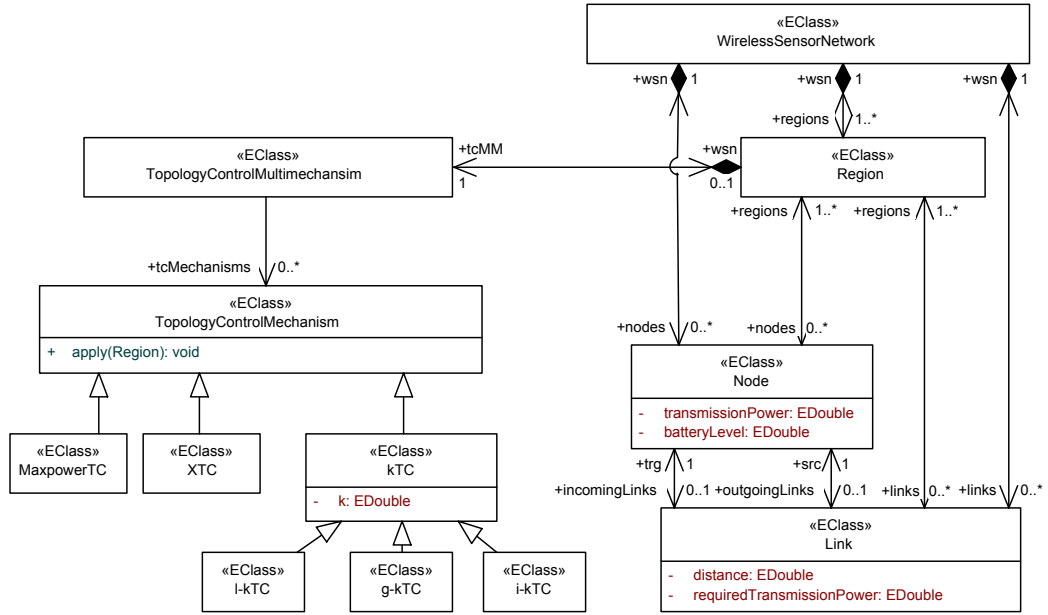


Figure 3: Metamodel of Wireless Sensor Networks

transmissionPower and batteryLevel and always belongs to one WirelessSensorNetwork and to at least one Region. Sensor nodes are interconnected by directed Links, having the real-valued attributes transmissionPower and distance. Additionally, two associations between Link and Node represent outgoing and incoming links.

The class TopologyControlMechanism represents a generic topology control mechanism. The method TopologyControlMechanism::apply(Region) runs the particular topology control algorithm on a given region of a WSN. A topology control mechanism typically works by determining and inactivating power-intensive links that are unnecessary for the operation of the WSN. In literature, hundreds of TC algorithms have been proposed; for conciseness, we list only a few of them in the metamodel:

- MaxpowerTC retains the complete—so-called maxpower—topology by inactivating no links at all<sup>4</sup>. While a maxpower topology is in most cases not the most energy-efficient one, it is relatively robust, e. g., against sudden nodes failures, due to its high degree of redundancy.
- XTC sorts the neighbors of each node by quality (e. g., by increasing distance) and removes all links whose endnode is already ‘covered’ by

<sup>4</sup>Indeed, MaxpowerTC instantiates the *Null Object* pattern [20].

higher-quality links [58].

- **kTC** reduces (unnecessary) redundancy in a topology by inactivating all links that are the longest link in some triangle [48]. The parameter  $k$  may be used to control the ‘aggressiveness’ of the algorithm: Some longest link in a triangle is removed only if it is additionally at least  $k$  times longer than the shortest link in the triangle.
- **g-kTC** and **l-kTC** [28, 51] are variants of **kTC** that take application-specific constraints into consideration. Generally speaking, **g-kTC** and **l-kTC** only remove links from a topology if the increase of the length of routing paths in the overlay stays below a given threshold.
- **i-kTC** [27] is an incremental variant of **kTC**. While **kTC**, **g-kTC**, and **l-kTC** tend to operate on whole topologies, **i-kTC** operates on change events of the topology.

The `WSNTopologyControlMultimechanism` has a number of registered `WSNTopologyControlMechanisms`, from which it may choose the most suitable one(s) at runtime.

The metamodel in Figure 3 presents a highly flexible approach to specifying WSNs: If the WSN consists of only one region, the same topology control multimechanism is active for the whole WSN, which is especially useful when applying topology control mechanisms that require non-local knowledge of the WSN topology (e. g., overlay routing paths in **g-kTC** and **l-kTC** [28, 51]). If the WSN consists of more than one region, in the extreme case, each region consists of exactly one node. This allows the topology control multimechanism to configure topology control to meet the characteristics of the local topology, e. g., using **kTC** in ‘stable’ regions and **MaxpowerTC** in regions with high mobility. As a result, this enables different mechanisms to coexist independently within one WSN.

### 5.1.3 Example 2: Location-Assisted Publish-Subscribe

As a second example, we consider the location-assisted publish-subscribe framework **Bypass** [38]. The corresponding metamodel of **Bypass** is depicted in Figure 4. A **Bypass** system consists of a **Server** and a number of **Clients**. A **Client** emits **Events** and may register for **Events** emitted by other **Clients** at the **Server**. A **SubscriptionInterest** determines the types of **Events** that a **Client** may register for: A **TopicBasedInterest** describes interesting **Events** by their topic, while a **LocationBasedInterest** describes interesting **Events** by a circular area (given as radius) around a certain position (given as latitude

## 5.1 Structural Modeling Based on Metamodels

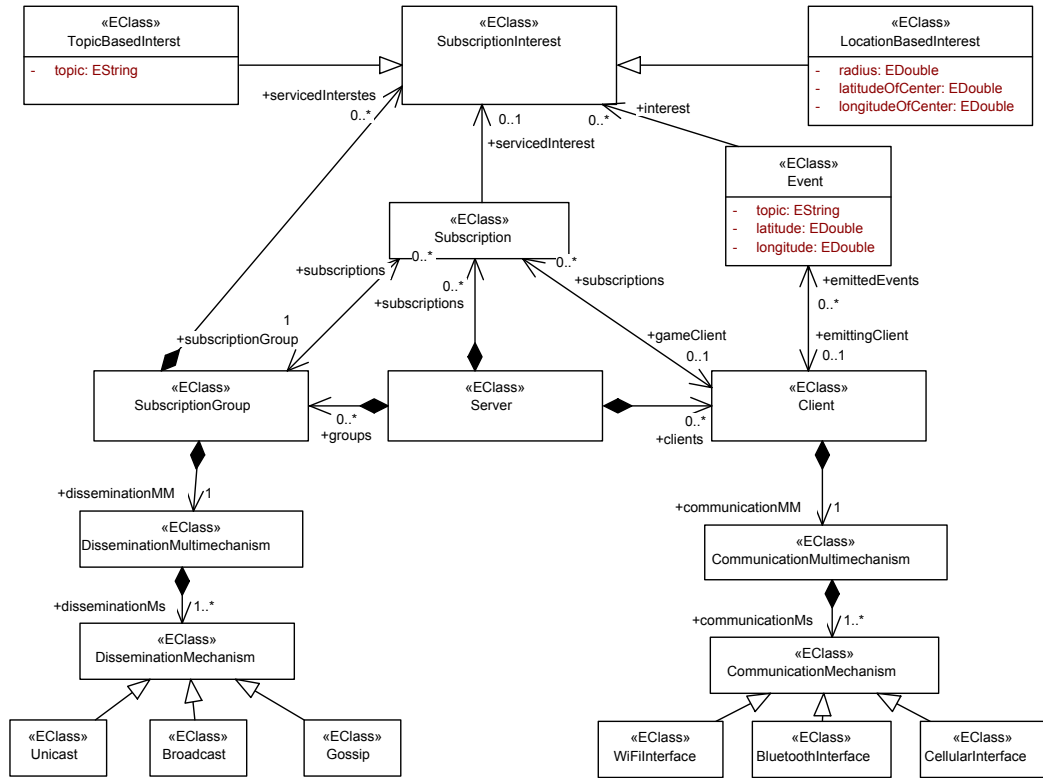


Figure 4: Metamodel of the location-assisted publish-subscribe system Bypass

and longitude). Each Client may be a member of zero or more SubscriptionGroups. A subscription group serves one or more SubscriptionInterests. We model two multimechanisms of Bypass: the DisseminationMultimechanism and the CommunicationMultimechanism.

The DisseminationMultimechanism wraps the method that determines how Events are distributed within one SubscriptionGroup. Possible DisseminationMechanisms are (i) Unicast, where an Event is delivered to each Client in a group, separately; (ii) Broadcast, where an Event is delivered to all Clients in a group at the same time; (iii) Gossip, where an Event is forwarded to a subset of a Client's neighbors.

The CommunicationMultimechanism hides the currently active CommunicationMechanism(s) of a Client. In scenarios with typical end-user mobile devices, we may at least identify the following communication interfaces, which may be (partially) active in parallel: WiFi-, Bluetooth-, and Cellular-Interface.



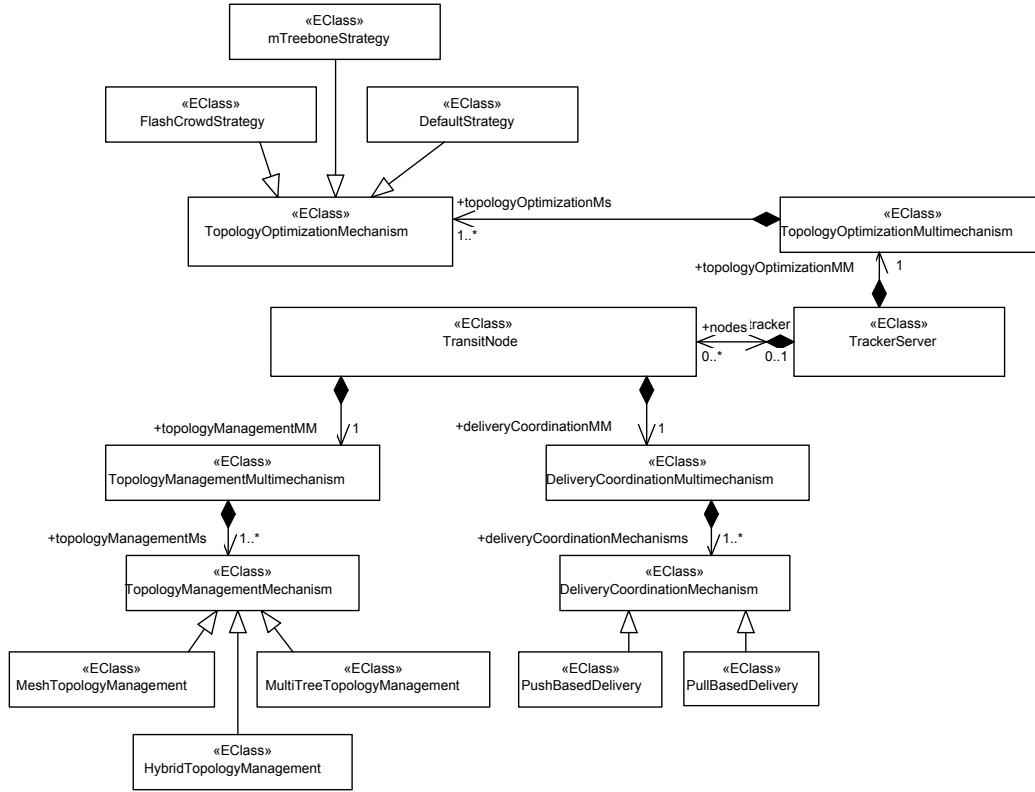


Figure 5: Metamodel of the adaptive video streaming system Transit

#### 5.1.4 Example 3: Adaptive Video Streaming System

As another example we consider the adaptive video streaming framework Transit [60, 42]. The corresponding metamodel of Transit is shown in Figure 5. In Transit, TransitNodes may register at a TrackerServer to receive a video stream. The delivery of the video stream is typically performed via multi-hop paths across several TransitNodes. The TransitNodes may build up different (logical) topologies. A TopologyManagementMechanism shapes the topology to either form a mesh (MeshTopologyManagement), a set of trees (MultiTreeTopologyManagement) or a combination of both (HybridTopologyManagement). On each TransitNode, a TopologyManagementMultimechanism hides the currently active TopologyManagementMechanism.

Additionally, a TransitNode may decide to receive/transmit the video stream in a pull-based and/or a push-based way, which is described by the DeliveryCoordinationMultimechanism with the two mechanisms PullBasedDelivery and PushBasedDelivery.

Finally, the `TopologyOptimizationMultimechanism` determines network-global strategies for handling special situations. For instance, the `Flash-CrowdStrategy` copes with high churn rates by restricting modifications of the topology to avoid premature optimizations. In contrast, the `mTreeboneStrategy` performs optimizations such as tree rebalancing.

**Summary** In this section, we showed how to specify the system structure and the placement of multimechanisms—on a network-/region-wide (WSN), group-wide (Bypass), or node-local (Transit) level—using metamodeling techniques. We provided metamodels of three representative examples of MAKI-enabled systems, exposing different levels of complexity. Still, these models already hint at certain limitations of pure metamodels: The interdependencies between multimechanisms cannot be described easily and in a sufficiently general way. Also, metamodels cannot be used to specify valid transitions between the mechanisms of one multimechanism. The following sections (especially Section 5.2) address these limitations and, thereby, complement the metamodeling approach described in this section. A common approach to describe the dynamic behavior of metamodels are so-called programmed graph transformations [47, 13] that are suitable to provide the implementation of class methods, which have been omitted here for conciseness, on the same abstraction levels as metamodels. A closer look at the three sample metamodels reveals that the topology of a communication system is an important aspect, which will be addressed separately in Section 6.4.

## 5.2 Describing System Configurations using Feature Models

In MAKI-enabled systems, the full variety of functionality is encapsulated in mechanisms, which may (i) be concurrently active at the same time, (ii) individually be replaced with other mechanisms at runtime, and (iii) expose conflicting requirements. In this section, we propose to apply techniques from the field of Software Product Line Engineering to tackle these challenges and provide a comprehensive specification of valid system configurations.

### 5.2.1 Specifying Variability of Dynamic Software Product Lines using FODA Feature Models

Feature models provide a comprehensive formalism for specifying commonality and variability among the different members of a family of similar (software) products organized in a software product line (SPL) [25]. A feature constitutes (i) a product characteristic, i. e., a system property relevant for some

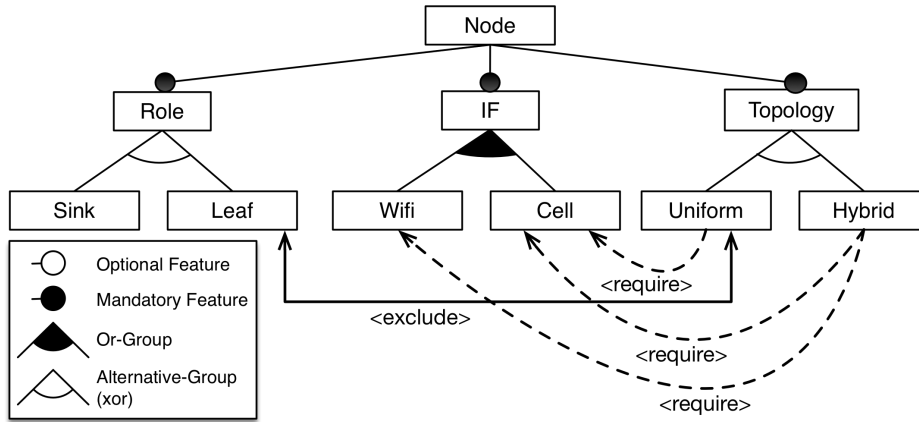


Figure 6: Feature model of Crater

stakeholder as identified during domain engineering, and (ii) a product configuration parameter for deriving stakeholder-specific product variants during application engineering [7]. A dynamic software product line (DSPL) enhances the SPL approach by allowing a product to be not only (pre-)configured once during application engineering, but rather by supporting flexible reconfigurations at runtime [6]. This enables a product implementation to dynamically evolve to meet continuously changing requirements [5].

As a running example, we consider the data dissemination protocol Crater [39] being part of a monitoring framework, which was designed using the MAKI paradigm as described in Section 2.3. Crater provides a multi-mechanism for which a selected mechanism depends on the current contextual situation. Relevant reconfiguration options of the MAKI-enabled system are captured as features that can be adapted at runtime. The constraints on feature combinations, as imposed by a feature model, restrict the configuration space of the DSPL to a subset of valid configurations. Figure 6 shows a sample feature model of Crater from the perspective of a single node in FODA notation [25].

The feature model organizes the supported features in a tree-like hierarchy. For instance, the feature Node decomposes into features Role, IF, and Topology. A single child feature is either *mandatory* or *optional* for its parent feature. For example, the feature Role constitutes mandatory core functionality to be part of every Crater variant. In addition, sets of child features may be collected in groups, where *or-groups* require at least one feature from that group to be present if its parent feature is present, whereas *alternative (xor) groups* require exactly one feature to be present. Finally, cross-tree edges

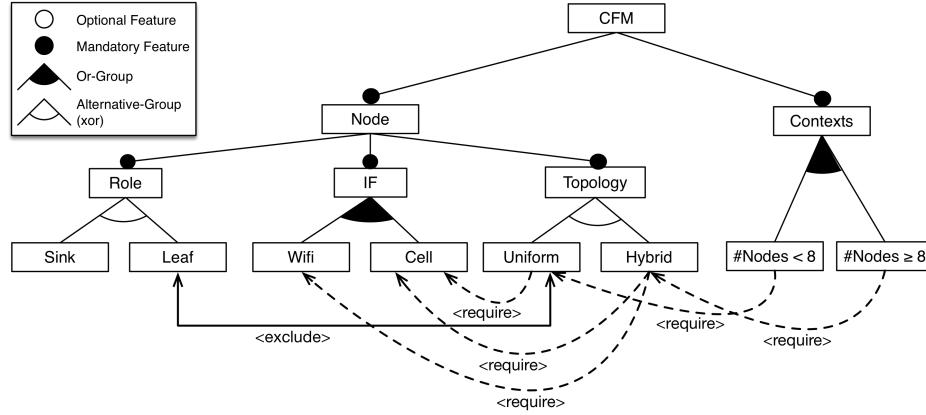


Figure 7: Context feature model of Crater

denote feature dependencies that crosscut hierarchies; for example, Leaf is incompatible with Uniform. A particular product configuration of a DSPL is obtained by binding all variability, i. e., by either selecting or deselecting every provided feature according to the given contextual situation.

### 5.2.2 Specifying Runtime Contexts using Context-Aware Feature Models

DSPL supports dynamical (de-)selection of features at runtime, depending on the requirements imposed by a contextual situation. To provide autonomous planning and execution of a reconfiguration at runtime, Saller et al. extended the DSPL variability specification given by a feature model with the requirements imposed by the contextual environment [44, 43]. The enhanced model builds the basis for a design-time pre-computation of appropriate reconfiguration behavior to be conducted by the implemented DSPL for the corresponding context changes emerging at runtime. The feature requirements of contexts are specified by *require* and *exclude* cross-tree constraints leading from the context model to the FM. As contexts represent distinct environmental states they are assigned by external events. The given FM of a DSPL is therefore extended with additional context information, again represented as a feature model over contexts, i. e., a context feature model.

Figure 7 shows the extension of the Crater DSPL by enriching its original FM (left-hand side) with a context model (right-hand side) resulting in a context-aware feature model (CFM). Here, all contexts are collected in an or-group which allows combinations of context features where always at least one context is to be active at runtime. Further constraints are specified by contexts requiring and/or excluding selected features. A reconfiguration is

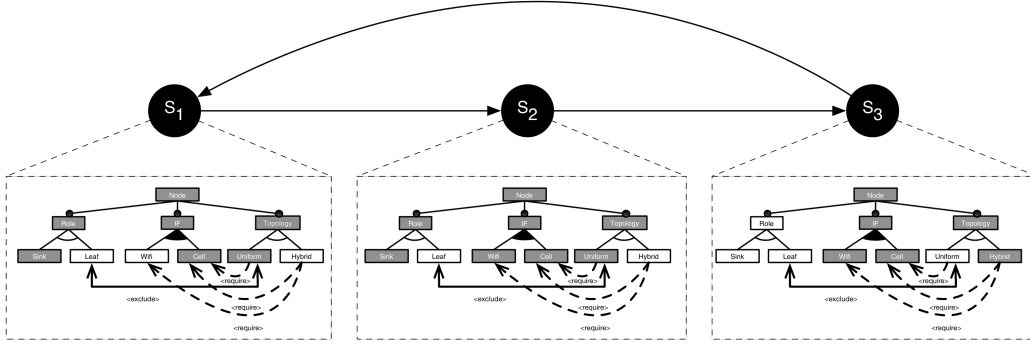


Figure 8: Reconfiguration model of handshake procedure in Crater

triggered by changes imposed by the currently active contexts.

### 5.2.3 Specifying Context-Aware Reconfiguration Behavior

Each valid configuration as specified by the CFM represents a state in the configuration state space. The reconfiguration between states establishes a reconfiguration sequence. However, not all reconfiguration sequences complying with the feature model are valid for a given DSPL. For instance, in Crater, the reconfiguration from a state where Uniform and Cell are active and WiFi is inactive to a state where Hybrid is active requires a handshake procedure. A reconfiguration is thus not directly possible without activating WiFi first with feature Uniform being selected. Therefore, the DSPL specification must be further augmented with a reconfiguration model restricting possible reconfiguration sequences precisely [32]. Saller et al. proposed to specify valid (partial) configurations as states and each potential reconfiguration as transitions in a transition system[44]. In [8, 23], automata-like specifications have been used for this purpose as well. Figure 8 depicts a sample transition system specifying the reconfiguration behavior of the CRATER DSPL.

As described in [43], based on this specification and enriched with an evolving probabilistic contextual model, further means of analysis such as state space reduction for resource constrained devices or off-line pre-computation for probable usage patterns were derived. An open issue is still the specification of reconfiguration behavior with reconfiguration automata consisting of states represented by partial feature configurations to tackle challenges such as state space explosion. Furthermore, we plan to generalize the notion of Context Feature Models, where contexts not only imply dependencies on system features but also potentially depend on them. This will further facilitate the specification of pervasive MAKI-enabled systems.

---

## 6 MAKI Transitions: Behavioral Aspects

### 6.1 Introduction

The goal of the MAKI collaborative research center is to contribute to a much more dynamic Future Internet. In this respect, the core concepts of mechanisms and transitions are introduced. The MAKI reference model introduced in Section 4 proposes a utility-based behavioral model for decisions about mechanism transitions. In other words, such transitions aim at maximizing the utility of the system by executing transitions between mechanisms at runtime. This chapter discusses a range of inroads made into the behavioral aspects of mechanism transitions. The contributions compiled here comply with the vision provided by the reference model, but take the existing Internet as a starting point and pursue evolutionary paths towards supporting dynamics.

Thereby, the researchers involved in this aspect of the MAKI research agenda addressed the problem from different viewpoints and, thereby, with different modeling approaches.

As an introduction, we will refer to three general approaches about decision making that are well known from the literature (cf. Figure 9). The approaches presented in the following fit very well with this classification: they can be associated with one or both of the first two categories and considered or integrated findings and general concepts from the third category. As Figure 9 indicates, the three wide-spread approaches are (i) to use an explicit utility function for deciding about transitions, (ii) to turn the problem into a goal-based specification, or (iii) to use rule based descriptions.

All these approaches have different advantages, application scenarios, and drawbacks, but have in common that they aim at increasing system performance. Basically, the three categories distinguish between different ways how the system derives the actual transition decision at *runtime*. Obviously, the developer of a rule-based description system has some utility function in mind that lead to the concrete rules. In the following, we illustrate the differences using concrete MAKI applications.

#### 6.1.1 Utility-Based Description

Utility functions are a widely used concept in adaptive systems, as they allow describing preferred configurations with a high level of abstraction [26]. Walsh et al. [56], for example, use utility functions to optimize the resource allocation in a dynamic, distributed environment. To derive the concrete adaptation decisions at runtime, these solutions require a detailed system model

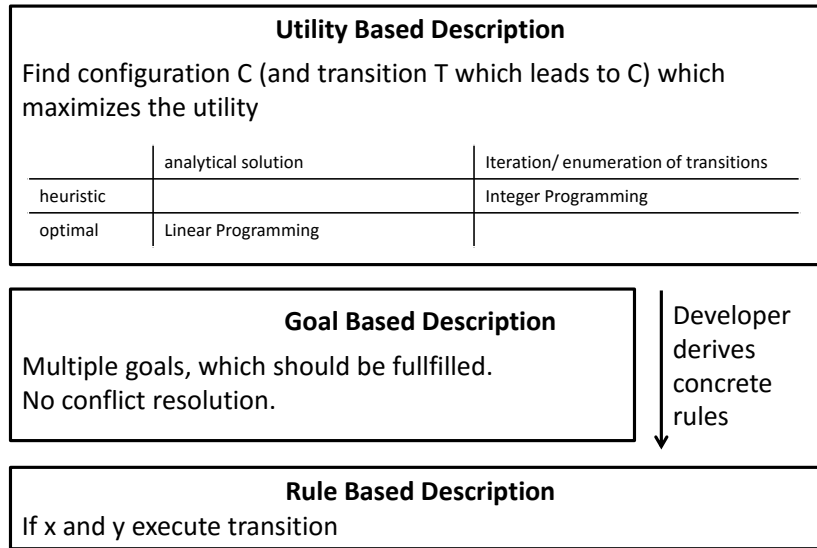


Figure 9: Classification of the transition decisions. The concrete transition at runtime is triggered based on different models.

to forecast the utility of different configurations and the costs of the adaptations [4].

A *Utility Based Description* searches for configurations or transitions which maximize the utility. Therefore, the systems uses an analytical models or iterates over a set of possible transitions at runtime. For each possible transition, the system evaluate the utility and finally choose the transition which leads to the maximum utility.

Utility based decisions play an important role in quite a number of MAKI projects. [11] is a good example where the concept of optimal mechanism combinations as promoted by MAKI was applied to a concrete Internet issue, namely video distributed using scaleable video codecs. Utility function were applied for that purpose. They are even applied in MAKI beyond decision making for transitions. For instance, [12] applies a utility function in order to maximize throughput in the wireless sensor network.

### 6.1.2 Goal Based

Goal based decision making has been pursued for many years in areas like behavioral agent research and in many knowledge based concepts. Following Kephart et al. [26], goal based specification of an adaptation logic described goals and constraints; the authors emphasize the fact that pure goal based approaches lack the possibility to specify conflict resolution strategies, e.g.,

in case multiple conflicting goals. This is one of the reasons why goal based approaches are often combined with rule based concepts. In our context, section describes a game theoretic approach to decision making for MAKI mechanism transitions that combines goal and rule based concepts.

### 6.1.3 Rule Based

An example for the derivation of concrete rules for a given utility function by the developer is kTC[48]. kTC is a simple local rule that removes the longest edge from triangles. By doing so, kTC achieves several goals, like an output topology with planarity and bounded node degree. However, these goals or a utility function are not explicitly encoded into the rule. Instead, it was evaluated during the development of the kTC rule that the rule implicitly leads to the aforementioned desired effects.

### 6.1.4 Rule Based plus Rule Learning

While rule based systems have well-known limits that make them just one out of three choices, recent developments in machine learning provide a path towards considerable sophistication. In particular, the automatic derivation of concrete rules for a given utility function, such as a rule-fuzzy learning proposed in [37] for video streaming, and the Fossa approach developed in the MAKI context [18] show that rule based systems along with rule learning constitute a very promising approach.

## 6.2 Behavioral Modeling Based on ECA Rules and Description Language

Event Condition Action (ECA) rules allow a developer to specify the events and conditions that trigger transition actions. The Fossa ECA Engine [17] is a concrete implementation of the ECA concept for transitions, which decouples the application from the transition logic.

The ECA rules have a human-readable representation which allows the developer to inspect and understand the transition logic. Additionally, they have a high expressiveness to support complex transition behavior.

ECA rules are triggered by *Events* which cause the evaluation of the *Condition*. If the condition evaluates to true, the *Action* leads to the execution of transitions. ECA rules can contain three types of events: i) application events that are pushed to the ECA engine (e.g., a *MessageReceived-Event*), ii)



updates and changes of important monitoring values (e.g., the current number of neighbors), or iii) clock events such as a periodic timer. Conditions are expressions consisting of monitoring values and event attributes. Monitoring values can be either pushed by the application to the ECA Engine or retrieved on demand. The ECA Engine executes simple optimizations such as constant folding and evaluation reordering on the expressions of the condition.

Actions execute the actual transitions which lead from one configuration to another. An ECA rule can contain multiple actions. This can be the simple change of a parameter, the exchange of a component following the life cycle specified in [19], or the addition or removal of a neighbor. Additional types and description approaches might be possible. Even the simple change of a parameter at runtime might require complex operations and have complex dependencies. The change of the *number of neighbors* in an overlay topology, for example, incurs additional network traffic for communication.

The ECA rule in Listing 1 illustrates the overall concept with an example. If the number of received messages in the last 30 seconds is greater than 5 and the number of hops a message is forwarded is less than 10, the application sets the number of neighbors to 10. The condition is evaluated after each *MessageReceivedEvent* and when the *NumberOfHops* changes.

```
1 on first match (count(MessageReceivedEvent, 30 s) > 5) and (←
    NumberOfHops < 10):
2 execute transition NumberOfNeighborTrans;
```

Listing 1: Example of an ECA rule.

To avoid oscillating transitions, conditions support temporal expressions, for example, “is the maximum number of neighbors during the last minute less than ten?” ( $\max(\text{NumberOfNeighbors}, 1 \text{ min}) < 10$ ). Additionally, rules can specify whether the action is executed (i) only the first time the condition matches, (ii) every time it matches, or (iii) every time it matches if not executed within the last  $t$  seconds. This enables fine-tuned transition behavior, which avoids oscillating transitions and overreactions. The Fossa ECA Engine is implemented efficiently with an event processing unit, which works as message broker between the incoming events and the corresponding ECA rules. The overall architecture of the Fossa ECA Engine ensures that only relevant monitoring values are collected, minimizing the monitoring overhead for the adaptivity.

The event and condition expressions are inspired by event processing languages such as ESPER Event Processing Language (EPL)<sup>5</sup> and Stream-SQL. There exists a multitude of other rule based description approaches for

<sup>5</sup><http://esper.codehaus.org/>

adaptive behavior, which influenced the design of the presented ECA rules: situation-actions rules of Dayal et al. [10], policies as described by Lobo et al. [31], condition-action rules [14], adaptation strategies and adaptation operators in the rainbow framework [21], adaptation rules as priority rules [15], policies [3], and action policies [26], to mention only some examples.

We extended the Fossa ECA Engine with an additional Fossa Learning Engine, which learns ECA rules for a given utility function [17]. This bridges the gap between a utility-based description of the desired adaptive behavior and a rule-based execution at runtime. Especially the genetic programming learner, as described in [18], leverages the clear syntax and semantics of the ECA rules as well as a model of the (possible) adaptive behavior, to learn ECA rules.

### 6.3 Behavioral Modeling Based on Min-Cost-Flow Optimization

In this subsection, we describe a min-cost-flow model to find optimal transitions. An optimal flow in this model corresponds to an optimal choice of transitions. An example is illustrated in Figure 10.

#### 6.3.1 Nodes

Each end user device and all MAKI-enabled devices are represented in our min cost flow model. In the min-cost-flow the set of nodes is  $V = D \cup T \cup C \cup R \cup \{\Omega\}$ .

**Devices  $D$ :** There is a node  $v_d$  representing each device and its current configuration  $c(v_d) \in C$ .

**Configurations  $C$ :** Let  $C$  be the set of all possible configurations. The set of valid configurations for a device  $d$  is given by  $C(d) \subseteq C$ . The current configuration of device  $d$  is denoted by  $c(d) \in C$ . In the model, there is a configuration node  $v_d \in C$  for each configuration  $c \in C$ .

**Transitions  $T$ :** We assume that every device can transition between any of its valid configurations. There is a node  $v_t \in T$  for each possible transition. The node with label  $a \rightarrow b$  with  $a, b \in C$  models the transition from configuration  $a$  to configuration  $b$ . We explicitly also have transition nodes  $a \rightarrow a$  for  $a \in C$  for not changing the configuration of a node.

**Restrictions  $R$ :** The model supports restrictions on the number of nodes attaining a certain configuration. Such a restriction  $r_j$  models the sum of devices attaining any of the configurations  $c \in C_j \subseteq C$  is at most  $k_j$ . Note that in this formulation any configuration may appear in at most one restriction.

All devices that are not restricted in any sense are connected to an unlimited restriction with sum at most  $\infty$ .

**Sink**  $\Omega$ : The sink  $\Omega$ .

### 6.3.2 Arcs

**Devices to Transitions:** For each device node  $v_d \in D$ , there is an arc  $(v_d, v_t)$  to all transition nodes  $v_t \in T$  with labels  $a \rightarrow b$  where  $a = c(v_d)$  is the current configuration of the device and  $b = C(v_d)$  is a possible configuration for the device.

**Transitions to Configurations:** For each transition node  $v_t \in T$ , there is an arc  $(v_t, v_c)$  to the corresponding target configuration of the transition, i.e., for  $v_t$  with label  $a \rightarrow b$  with  $a, b \in C$ , node  $v_c$  corresponds to configuration  $b$ .

**Configurations to Restrictions:** For each configuration node  $v_c \in C$ , corresponding to configuration  $c \in C$  we have an arc  $(v_c, v_{r_j})$  to the (unique) restriction node  $v_{r_j}$  containing  $c$ .

**Restrictions to the Sink:** For each restriction node  $v_{r_j} \in R$  there is an arc  $(v_{r_j}, \Omega)$  to the sink.

### 6.3.3 Costs

There are two different types of arc costs. The first type is the transition cost, which is the cost for implementing a transition, including costs for configuration changes measured in resource consumption, downtimes, intermediate drop in user experience, etc. The second type is the operation cost, which is the cost of operation within a certain configuration. The operation cost does not only contain costs (e.g., resource consumption) but also benefits (e.g., quality of experience). In the min cost flow model, these costs are modeled as follows:

**Transition costs**  $\delta$ : on all arcs from nodes in  $T$  to nodes in  $C$ .

**Operation costs**  $\gamma$ : on all arcs entering  $\Omega$ .

### 6.3.4 Flow Balance

Only the device nodes and the sink have non-zero balance. All the device nodes are supply nodes with a balance of  $+1$ , whereas the sink is a demand node with balance  $-|D|$ .

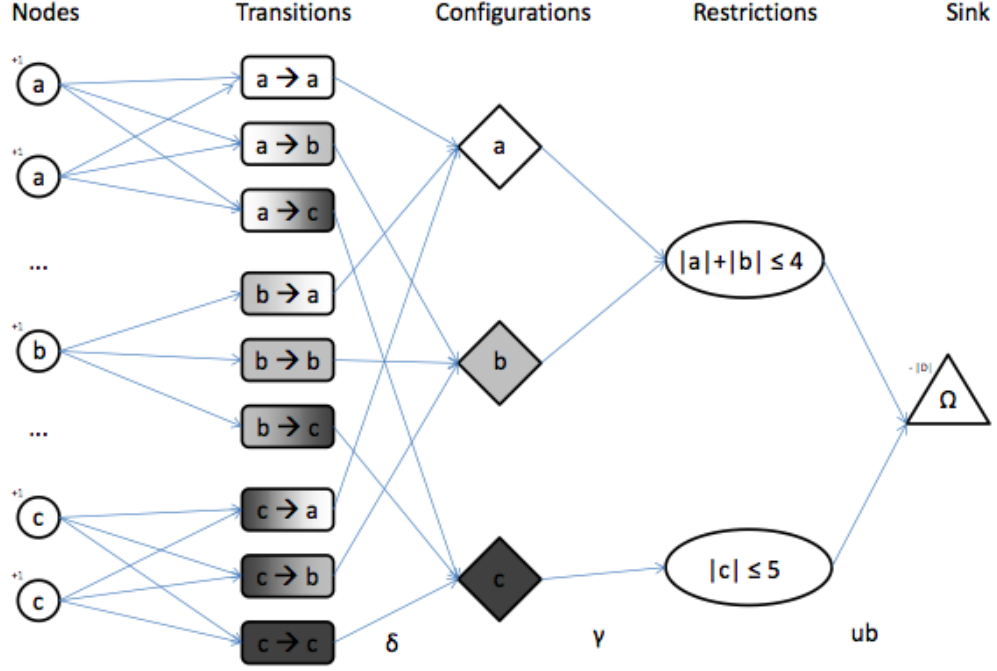


Figure 10: Min-cost-flow model

### 6.3.5 Bounds on the Flow

Only the arcs leaving the restriction nodes  $r_j$  have an upper bound on the flow value. This upper bound is exactly the value  $k_j$  of restriction  $r_j$  limiting the sum of all devices with configurations in  $C_j$ .

### 6.3.6 Optimization

To find an optimal set of transitions we first build the network according to the current system state. Next, an optimal flow is calculated using a standard solver for min-cost-flow problems. Finally, we determine the optimal transitions as follows. In an optimal result, for each device  $d$ , exactly one edge  $(v_d, v_t)$  has non-zero flow, more precisely a flow value of one. This arc determines the optimal transition  $t$  for device  $d$ .

### 6.3.7 Limitations and benefits

Unfortunately, the model has a limitation in expressiveness regarding constraints: Each configuration can only be part of at most one restriction. This is sufficient to model many real-world restrictions like limiting the number of devices in the same configuration or a list of configuration, e.g., due to a limit on the capacity of assigned resources in that configuration(s) (router, cell-phone tower, or similar). However, it does not allow for a more complex set of restrictions with more than one restriction on any of the configurations.

Although the number of configurations may become quite large, the model allows for quick optimization as the formulation results in layered graphs. Even huge instances can be solved to optimality quickly.

## 6.4 Modeling Topologies under Transitions

The topology of a communication system describes its inherent structure. Typically, a topology is modeled as a graph  $G = (V, E)$ , where the node set  $V$  and the edge set  $E$  denote the networked devices and the communication channels, respectively. We distinguish two types of topologies: physical topologies and logical topologies. An example of a physical topology is a wireless ad-hoc network, where an edge  $(u, v) \in E$  indicates that the incident nodes  $u, v$  are within transmission range of each other. On the other hand, an example of a logical topology are peer-to-peer overlays, where edges indicate the communication pattern of the overlay.

Both physical and logical topologies may be adapted at runtime. From a modeling perspective, this means that the graph  $G$  is modified. Typically, nodes have a certain degree of freedom when selecting their neighbors. For example, a node in a wireless network may select its neighborhood from the set of nodes that are contained in the maximum transmission range of the node [51]. In other words, the edge set  $E$  is modified and, thereby, topology-specific side constraints are considered. Moreover, the set of nodes  $V$  may be modified. In the example of a sensor network, a node may temporarily switch to a sleep mode in order to disable its wireless transmitter, resulting in a smaller energy consumption [1]. Switching between the different modes leads to the node vanishing from and re-appearing to  $V$ .

Using a common model like graph theory for topology adaptations is beneficial for two main reasons. First, different adaptation concepts and algorithms can be jointly analyzed and compared, independent from a specific topology mechanism. Second, a common model allows for providing generic and replaceable adaptation components, which can be re-used for different (related) topology mechanisms. An example of such an adaptation compo-

nent is a component that adapts the graph toward a certain distribution of target patterns. As exemplarily demonstrated above, most topology adaptations may be modeled as edge modifications or node modifications of a graph  $G$ , i.e., the edge set  $E$  or the node set  $V$  may be modified. In order to support its adaptation by a generic component, a mechanism needs to provide an interface for these graph operations and map these operations to mechanism-specific operations, like connecting to or disconnecting from a specific neighbor. Moreover, it must be ensured that inherent properties of the mechanism's topology are never violated. For this purpose, the interface may for example allow for requesting graph constraints. Another possibility would be an interface that, at each point in time, provides a set of currently allowed graph modifications.

In the context of a MAKI-enabled system, we distinguish two types of adapting the mechanism's topology at runtime: mechanism-internal adaptation and transition-enforced adaptation.

Mechanism-internal adaptation describes the modification of a topology within a mechanism in order to optimize given optimization criteria. This adaptation is conducted continuously and independent of a transition. A multitude of mechanism-internal adaptations exist. A typical example is topology control, where physical communication links are removed from wireless networks, leading to energy savings [48, 51]. Other examples are mechanism-internal adaptation for improved event dissemination in peer-to-peer networks [30], improved connectivity in ad-hoc networks [9], or streaming tree balancing [57]. A wide-spread approach to conduct mechanism-internal adaptation are local algorithms. In accordance with this section, local algorithms are often defined and applied on graphs [53]. In a local algorithm, each node  $v \in V$  has a local view  $G_{\mathcal{L}}(v) \subseteq G$  of small size. If each node adapts its local view  $G_{\mathcal{L}}(v)$  in an appropriate way, a global effect on  $G$  can be achieved cooperatively. An example of a local algorithm is the aforementioned kTC algorithm [48], where each node has a local view of two hops and applies a simple rule to remove edges from  $G_{\mathcal{L}}(v)$ .

Transition-enforced adaptation describes the modification of a topology in the context of a transition. In order to motivate such an adaptation, we consider two well-known peer-to-peer search overlays: Gnutella [40] and BubbleStorm [54]. Recently, we have shown that transitions between Gnutella and BubbleStorm are possible [16]. BubbleStorm and Gnutella exhibit inherently different topologies: While BubbleStorm relies on a random graph to give probabilistic guarantees, Gnutella has characteristics of a power-law structure. As becomes easily evident, conducting a transition between the mechanisms BubbleStorm and Gnutella requires that the topology is adapted accordingly.

Conducting a transition-enforced adaptation is challenging because several requirements need to be fulfilled. In general, a topology transformation from a graph  $G_A$  to a graph  $G_B$  should be conducted in an efficient way with respect to time and communication overhead. For this reason, if possible, the existing topology  $G_A$  should be modified rather than constructing a new topology from scratch. Based on application-specific needs, we can think of different additional requirements for the transition-enforced topology adaptation. For example, considering the transitions between search overlays, as considered in the example above, search queries still need to be answered successfully during the transition. Such specific characteristics need to be considered during the modification of a topology.

For the aforementioned transition between peer-to-peer search overlays, a transition-enforced adaptation can be performed for example by stopping the source overlay and constructing a new overlay by using the join procedure provided by the mechanism under transition [16]. While this approach is straightforward, it exhibits the disadvantage that a complete new overlay is constructed, a costly and time-consuming operation.

A possible approach to be considered in the future is to develop concepts and algorithms for efficient transformations between specific types of graphs. If no suited direct transformation for a specific pair of types of graphs exists, such transformations could be combined smartly. For example, typical topology control algorithms for wireless networks rely on a Unit Disk Graph as base topology. By providing transformations from different topologies back to this base topology (which essentially only requires re-adding all edges that were removed from the Unit Disk Graph before), transitions between different wireless topologies could be realized by combining the transformation to the base topology with an additional transformation to the target topology. Following a generic model of topology adaptation as proposed above, such concepts and algorithms could potentially be re-used in different implementations and even application domains.

## 6.5 Modeling Transitions with Game Theory

In this subsection, we describe how game theoretical methods can be applied to model autonomous behavior in MAKI systems. First, we describe why in general decentralized algorithms are of high importance for MAKI systems. Then, we explain why especially game-theoretic decentralized algorithms can be well applied to MAKI systems.

### 6.5.1 Importance of Decentralized Algorithms for MAKI Systems

In order to control or optimize a network by a centralized algorithm, all information about the network originating from different nodes must be collected at a central unit. In large networks, this process consumes high amounts of resources, such as time and energy needed for coordination [55]. Compared to that, less resources have to be spent when nodes act autonomously based on local information. Therefore, decentralized algorithms are suitable, in which nodes rationally optimize their own parameters based on local knowledge. In this part, we consider networks, in which each of the network's nodes has its own transition control logic. Given a certain multi-mechanism, each node triggers its own transitions between equivalent mechanisms individually, depending on the node's benefits and costs of each transition. Examples for such networks are wireless Ad Hoc networks, in which each node represents a mobile device of some end user. Below, we give two exemplary multi-mechanisms in such networks that depend on the individual decisions of each node.

**Example 1: Multi-hop Broadcasting for Data Dissemination** Consider a data dissemination scenario in an Ad Hoc network, in which a video available at a source node is distributed in a multi-hop manner to a number of nodes [33, 34]. For such a scenario, some forwarding nodes must forward the source's data for other receiving nodes. Upon receiving the video from a forwarding node, the receiving node should re-transmit the video to nodes in its proximity. Depending on the number of receiving nodes that a forwarding node has, the transmission at the forwarding node can be either by unicast or multicast, when a forwarding node has one or several receiving nodes, respectively. Performing a transition between the two mechanisms "Unicast" and "Multicast" and deciding about its receiving nodes, the node can adjust its transmit power. Since a node's current mechanism determines which links are active, the topology of the network depends on all nodes' decisions. While from a network's point of view, a topology that minimizes the overall energy consumption might be desired, such a topology might be less beneficial for single nodes. Hence, the nodes' local decisions usually lead to a suboptimal overall outcome.

**Example 2: Task Offloading Mode** Consider a computation offloading scenario, in which a number of mobile nodes possess computation tasks [2, 35]. A node can either process its task locally, or send it to a resourceful server via a multi-hop route. Performing a transition between the two mechanisms "Pro-



cess Locally" and "Offload to Server", the node might reduce its own energy consumption. At the same time, in case the node decides to offload its tasks, other nodes are required to forward the data to the server, which increases their energy consumption. In this scenario, the network topology affects the nodes' decisions, which in turn affect the load on the network. Again, a globally optimal configuration, e.g. minimizing the overall energy consumption, might be in conflict with individual nodes' benefits.

The examples illustrate that firstly, benefits and costs for transitions are individual to each node. This can be captured by introducing a local utility function for each node in the network (c.f. MAKI specific requirement (M4)), which trades the expected benefit of a transition off against the expected transition costs. Secondly, when globally optimal transitions are not enforced by a central planner since nodes execute transitions autonomously, the nodes' decisions might not lead to a desired overall state of the network. Both observations are naturally included in the notion of non-cooperative game theory.

### 6.5.2 A Short Overview of Non-Cooperative Game Theory

Non-cooperative game theory investigates decision processes of players with potentially conflicting interests [36]. Each player has to take his own decisions, taking into account the possible decisions of other players and the effect of the other players' decisions on his own objective. In this way, the players' decisions are coupled and determine the outcome of the decision process. Each individual player's objective is described by a utility function, which is a mapping from all players' decisions to the resulting payoff for the individual player. Assuming rationality and selfishness, the goal of each player is the maximization of his own utility. The fact that decisions are taken autonomously without relying on a central planner, makes game theory a suitable method to model networks in a decentralized way [22, 29].

**Nash Equilibrium – A Solution Concept** A solution concept of a game is a Nash Equilibrium, a state in which no player can improve his utility by unilaterally changing his strategy given the current strategies of other players [36]. A Nash Equilibrium does not necessarily correspond to the best outcome of the game, but it is a state at which a player has no incentive to change his strategy. The goal of *mechanism design*, a special field in game theory, is to design the rules of a game in a way to obtain a certain outcome when self-interested players get involved in competitive situations [50].

**Price of Anarchy (PoA)** The Price of Anarchy (POA) is a measure to study the efficiency of a game's outcome. Due to selfish behavior of the players in non-cooperative games, the Nash equilibrium of the game is often inefficient compared to the globally optimal solution. PoA is defined as the ratio between the worst possible NE and the global optimum of the problem achieved by a central authority [50]. In other words, PoA shows the maximum possible loss in performance when a problem is solved in a game-theoretic decentralized way instead of a centralized optimization.

### 6.5.3 Applying Game Theory to MAKI Systems

**Analyzing the System and Designing Incentives** Autonomous behavior in MAKI systems can be modeled with game theory by considering each node in the network as a selfish player. The possible MAKI mechanisms of a node correspond to the actions of the player. Moreover, the local utility function of a node, which trades the expected benefit of a mechanism off against its expected cost, represents the game theoretic utility of the action. In this way, a MAKI transition at a node corresponds to a (game theoretical) change of the action of a player. Since the utility of a network's node is usually affected by the decisions of other nodes and since interests of nodes can be conflicting, such a translation of a MAKI-system into a non-cooperative game theoretic model is useful.

As an illustration, consider Example 1 from Section 6.5.1. For a single node in the network, sending the source's data to several child nodes using multihop broadcast might not be in the interest of a forwarding node, since this requires energy consumption. On the other hand, it might be beneficial for the receiving nodes since their direct link to the source might require a high number of retransmissions, delay and energy consumption. Therefore, different nodes may have conflicting interest in a multihop broadcast scenario. Let  $u_i(a_i, \mathbf{a}_{-i})$  represent the utility function of player  $i$  which is a function of the action  $a_i$  of player  $i$ , and the actions  $\mathbf{a}_{-i}$  of other players. Suppose the goal is the minimization of energy consumption in the network. Then, the utility function at the nodes must be defined as a function of energy. If we define a cost at every node equal to the energy which is consumed at a nodes' respective transmitter, the node chooses a transmitting node for receiving data which leads to a lower cost for himself. Therefore, by maximizing the utility (minimizing the cost) at a node, the energy consumption in the network is minimized in a fully decentralized way [34]. Similar assumption can be made for Example 2 of Section 6.5.1. For computation offloading, in order to minimize the energy consumption in a network, a local utility function should consist of both the device's internal energy consumption and external

energy consumption for transmission and offloading to the cloud [2, 35].

After modeling nodes in a network as players in a non-cooperative game, based on the individual nodes' utilities and their coupling effects, the properties of the game (c.f. Section 6.5.2), such as the type of game or the existence and uniqueness of Nash Equilibria can be analyzed. For example, it can be identified, which transitions the nodes are likely to perform and which configurations of mechanisms are likely outcomes of the transition process. Additionally, since Nash Equilibria may not lead to the globally optimal solution of a game (c.f., Section 6.5.2), also the transition process of nodes in the MAKI system might not lead to the globally optimal solution. Hence, from a game theoretic perspective, it is crucial to investigate how the game has to be adapted, such that it can reach a better equilibrium point. Translating this to MAKI systems, this means that incentive mechanisms (such as pricing or reputation systems) have to be developed which persuade nodes to make transitions that are locally less beneficial, but globally beneficial.

**Incorporating Cross-Layer Utilities** A utility function represents the trade-off between the benefits and the costs of players in a game. In communication networks, the benefits and costs of mechanisms depend on several parameters from different communication layers at the same time. Often, utility functions only consider the parameters from a single layer, either in the underlay (e.g. energy consumption, network load, delay) or overlay (e.g. video quality, application requirements). A model in which the utility function considers parameters from different communication layers is called a cross-layer model. In [59], a cross-layer model is presented for a multihop video dissemination scenario. In this scenario, the proposed utility function of a node is composed of the quality of experience of the node as benefit and the energy that the node consumes in the network as cost. Based on the contribution of a node in terms of energy consumption and forwarding video for others, the node receives a higher quality of the video from its respective transmitter. Here, receiving the video with higher quality is an incentive for the nodes to contribute more in the network.

Designing a cross-layer utility is more challenging than a single-layer utility. It requires that the designers have information about both underlay and overlay parameters affecting the network performance. For instance, consider the wireless channel gain in the physical layer and the delay of routing protocol in the network layer. For this specific example, the parameters in the physical layer usually change in the scale of milliseconds while for the network layer this delay might be in the scale of minutes. A designer of cross-layer models must take this issue into account. Although designing a cross-

---

layer framework is more difficult than a single layer framework, the overall system outcome with a cross-layer model is more satisfying since it covers a broader range of parameters [49].

## 7 Conclusions

The present whitepaper motivates the need for a revision of the OSI reference model and of the governing principles of the Internet. The proposed revision aims at making the Internet much more adaptive, enabling fast and many-fold switching between alternative approaches for the sake of best possible performance and user experience. The ever increasing dynamics w.r.t. user behavior and load, mobility patterns, and crowd phenomena motivate the quest for much better support for dynamicity. In this respect, the *MAKI reference model* was introduced with an emphasis on *mechanism transitions*, the core concept for better dynamics.

The whitepaper also introduced *concrete* modeling and specification approaches that make the MAKI reference model (partly) available to developers, such that they can realize MAKI-compliant communication systems, i. e., Future Internet components that support mechanism transitions and the implementation of (“M-A-P-E-inspired”) control cycles which drive transitions and make the network highly adaptable.

As to the structural model, a useful metamodeling approach was introduced, extending and leveraging concepts known from feature modeling. For coping with large combinations of mechanisms, an approach based on feature models was introduced.

As to the behavioral model, quite a number of approaches were described, mostly rule-based and goal-based approaches. One rule-based approach was extended towards machine learning of optimization rules. The examples included approaches towards modeling the control cycle that drives transitions. However, it became clear that more sophisticated and holistic control cycles remain an open issue. In a nutshell, such future approaches will have to consider function based descriptions of major transition drivers (performance functions, cost functions, utility functions and the like) where the (rule and goal based) approaches presented for MAKI use expressions where, e. g., performance characteristics and utility functions are combined in a single term. Given the increasing dynamics of Internet users and their usage patterns, the MAKI approach towards a much more dynamic Internet remains an important promise for the Future Internet.

## A References

- [1] A Survey on Distributed Topology Control Techniques for Extending the Lifetime of Battery Powered Wireless Sensor Networks. *IEEE Communications Surveys & Tutorials*, 15(1):121–144, jan 2013.
- [2] H. Al-Shatri, S. Müller, and A. Klein. Distributed algorithm for energy efficient multi-hop computation offloading. In *IEEE International Conference on Communications [accepted]*, pages 1–6, 2016. in press.
- [3] R.J. Anthony. A policy-definition language and prototype implementation library for policy-based autonomic systems. In *IEEE Autonomic Computing (ICAC)*, pages 265–276, June 2006.
- [4] Matthias Becker, Markus Luckey, and Steffen Becker. Model-driven performance engineering of self-adaptive systems: a survey. In *Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures*, pages 117–122. ACM, 2012.
- [5] Nelly Bencomo, Svein Hallsteinsen, and Eduardo Santana de Almeida. A view of the dynamic software product line landscape. *Computer*, 45(10):36–41, 2012.
- [6] Nelly Bencomo, Peter Sawyer, Gordon S Blair, and Paul Grace. Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems. In *SPLC (2)*, pages 23–32, 2008.
- [7] Krzysztof Czarnecki and Ulrich W Eisenecker. *Generative programming*. Addison-Wesley Professional, 2000.
- [8] Ferruccio Damiani and Ina Schaefer. Dynamic delta-oriented programming. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, page 34. ACM, 2011.
- [9] Shantanu Das, Hai Liu, Amiya Nayak, and Ivan Stojmenović. A localized algorithm for bi-connectivity of connected mobile robots. *Telecommunication Systems*, 40(3–4):129–140, 2009.
- [10] Umeshwar Dayal, Alejandro P Buchmann, and Dennis R McCarthy. Rules are objects too: a knowledge model for an active, object-oriented database system. In *Advances in Object-Oriented Database Systems*, pages 129–143. Springer, 1988.

- 
- [11] M. Fasil, H. Al-Shatri, S. Wilk, and A. Klein. Application-aware cross-layer framework: Video content distribution in wireless multihop networks. In *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2015 IEEE 26th Annual International Symposium on*, pages 1088–1093, Aug 2015.
  - [12] M. Fasil, A. Kuehne, and A. Klein. Node virtualization and network coding: Optimizing data rate in wireless multicast. In *Wireless Communications Systems (ISWCS), 2014 11th International Symposium on*, pages 573–578, Aug 2014.
  - [13] Thorsten Fischer, Jörg Niere, Lars Torunski, and Albert Zündorf. Story diagrams: A new graph rewrite language based on the Unified Modeling Language. In *Proc. of the International Workshop on Theory and Application of Graph Transformation*, pages 296–309. Springer, 1998.
  - [14] Franck Fleurey, Vegard Dehlen, Nelly Bencomo, Brice Morin, and Jean-Marc Jézéquel. Modeling and validating dynamic adaptation. In Michel R.V. Chaudron, editor, *Models in Software Engineering*, volume 5421 of *Lecture Notes in Computer Science*, pages 97–108. Springer Berlin Heidelberg, 2009.
  - [15] Franck Fleurey and Arnor Solberg. A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. In *Model Driven Engineering Languages and Systems*, pages 606–621. Springer, 2009.
  - [16] Alexander Frömmgen, Stefan Haas, Michael Stein, Robert Rehner, Alejandro Buchmann, and Max Mühlhäuser. Always the best: Executing transitions between search overlays. In *Proceedings of the 2015 European Conference on Software Architecture Workshops, ECSAW '15*, pages 8:1–8:4, New York, NY, USA, 2015. ACM.
  - [17] Alexander Frömmgen, Robert Rehner, Max Lehn, and Alejandro Buchmann. Fossa: Learning ECA Rules for Adaptive Distributed Systems. In *ICAC*, pages 1–4, Jul. 2015.
  - [18] Alexander Frömmgen, Robert Rehner, Max Lehn, and Alejandro Buchmann. Fossa: Using Genetic Programming to Learn ECA Rules for Adaptive Networking Applications. In *LCN*, pages 1–4, Oct. 2015.
  - [19] Alexander Frömmgen, Björn Richerzhagen, Julius Rückert, David Hausherr, Ralf Steinmetz, and Alejandro Buchmann. Towards the Description and Execution of Transitions in Networked Systems. In *AIMS*, pages 17–29, Jun. 2015.

- [20] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.
- [21] D. Garlan, Shang-Wen Cheng, An-Cheng Huang, B. Schmerl, and P. Steenkiste. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, Oct 2004.
- [22] Zhu Han, Dusit Niyato, Walid Saad, Tamer Başar, and Are Hjørungnes. *Game Theory in Wireless and Communication Networks*. Cambridge University Press, 2011.
- [23] Michiel Helvensteijn. Dynamic delta modeling. In *Proceedings of the 16th International Software Product Line Conference-Volume 2*, pages 127–134. ACM, 2012.
- [24] Information Technology – Object Management Group (OMG). Meta Object Facility (MOF) Core. Technical report, ISO, 2014.
- [25] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University, 1990.
- [26] J.O. Kephart and R. Das. Achieving Self-Management via Utility Functions. *Internet Computing, IEEE*, 11(1):40–48, Jan 2007.
- [27] Roland Kluge, Gergely Varró, and Andy Schürr. A methodology for designing dynamic topology control algorithms via graph transformation. In Dimitris Kolovos and Manuel Wimmer, editors, *Theory and Practice of Model Transformations*, volume 9152 of *Lecture Notes in Computer Science*, pages 199–213. Springer International Publishing, 2015.
- [28] Géza Kulcsár, Michael Stein, Immanuel Schweizer, Gergely Varró, Max Mühlhäuser, and Andy Schürr. Rapid prototyping of topology control algorithms by graph transformation. In *Proc. of the 8th Int. Workshop on Graph-Based Tools*, volume 68 of *ECEASST*, 2014.
- [29] Samson Lasaulce and Hamidou Tembine. *Game Theory and Learning for Wireless Networks, Fundamentals and Applications*. Elsevier, 2011.
- [30] Max Lehn, Robert Rehner, and Alejandro Buchmann. Distributed Optimization of Event Dissemination Exploiting Interest Clustering. In *Proceedings of the Conference on Local Computer Networks (LCN)*, pages 328–331, 2013.

- 
- [31] Jorge Lobo, R Bhatia, and S Naqvi. Apolicy description language. In *AAAI Artificial Intelligence*, pages 291–298, 1999.
- [32] Malte Lochau, Johannes Bürdek, Stefan Hölzle, and Andy Schürr. Specification and automated validation of staged reconfiguration processes for dynamic software product lines. *Software & Systems Modeling*, pages 1–28, 2015.
- [33] Mahdi Mousavi, Hussein Al-Shatri, Hong Quy Le, Alexander Kuehne, Matthias Wichtlhuber, David Hausheer, and Anja Klein. Game-based multi-hop broadcast including power control and MRC in wireless networks. In *26th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–6, August 2015.
- [34] Mahdi Mousavi, Hussein Al-Shatri, Matthias Wichtlhuber, David Hausheer, and Anja Klein. Energy-efficient data dissemination in ad hoc networks: Mechanism design with potential game. In *12th IEEE International Symposium on Wireless Communication Systems*, pages 1–5, August 2015.
- [35] Sabrina Müller, Hussein Al-Shatri, Matthias Wichtlhuber, David Hausheer, and Anja Klein. Computation offloading in wireless multi-hop networks: Energy minimization via multi-dimensional knapsack problem. In *26th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–6, August 2015.
- [36] Martin Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.
- [37] Guillaume F Remy, Hassan B Kazemian, and Phil Picton. Application of neural-fuzzy controller for streaming video over bluetooth 1.2. *Neural Computing and Applications*, 20(6):879–887, 2011.
- [38] Björn Richerzhagen, Dominik Stingl, Ronny Hans, Christian Gross, and Ralf Steinmetz. Bypassing the cloud: Peer-assisted event dissemination for augmented reality games. In *Peer-to-Peer Computing (P2P), 14-th IEEE International Conference on*, pages 1–10, Sept 2014.
- [39] Nils Richerzhagen, Dominik Stingl, Björn Richerzhagen, Andreas Maathe, and Ralf Steinmetz. Adaptive monitoring for mobile networks in challenging environments. In *The 24th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–8, Aug 2015.
-



- [40] Matei Ripeanu. Peer-to-Peer Architecture Case Study: Gnutella Network. In *Proceedings of the International Conference on Peer-to-Peer Computing (P2P)*, pages 99–100, 2001.
- [41] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1: Foundations. World Scientific, 1997.
- [42] Julius Rückert, Jeremias Blendin, and David Hausheer. Software-defined multicast for over-the-top and overlay-based live streaming in isp networks. *Journal of Network and Systems Management*, 23(2):280–308, 2015.
- [43] Karsten Saller. *Model-Based Runtime Adaptation of Resource Constrained Devices*. PhD thesis, TU Darmstadt, 2015.
- [44] Karsten Saller, Malte Lochau, and Ingo Reimund. Context-aware dspls: model-based runtime adaptation for resource-constrained systems. In *Proceedings of the 17th International Software Product Line Conference co-located workshops*, pages 106–113. ACM, 2013.
- [45] Volker Gruhn Sami Beydeda, Matthias Book. *Model-driven Software Development*, volume 15. Springer, 2005.
- [46] Paolo Santi. Topology control in wireless ad hoc and sensor networks. *ACM computing surveys (CSUR)*, 37(2):164–194, 2005.
- [47] Andy Schürr. In [41], chapter Programmed Graph Replacement Systems, pages 479–546. World Scientific, 1997.
- [48] Immanuel Schweizer, Michael Wagner, Dirk Bradler, Max Mühlhäuser, and Thorsten Strufe. kTC – Robust and adaptive wireless ad-hoc topology control. In *Proc. of the 21st International Conference on Computer Communications and Networks*, 2012.
- [49] S. Shakkottai, T.S. Rappaport, and P.C. Karlsson. Cross-layer design for wireless networks. *IEEE Communications Magazine*, 41(10):74–80, Oct 2003.
- [50] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008.

- 
- [51] Michael Stein, Géza Kulcsár, Immanuel Schweizer, Gergely Varró, Andy Schürr, and Max Mühlhäuser. Topology Control with Application Constraints. In *Proceedings of the Local Computer Networks Conference (LCN)*, pages 1–4, 2015.
- [52] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.
- [53] Jukka Suomela. Survey of Local Algorithms. *ACM Computing Surveys*, 45(2):24:1–24:40, 2013.
- [54] Wesley W. Terpstra, Jussi Kangasharju, Christof Leng, and Alejandro P. Buchmann. BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search. In *Proceedings of the 2007 ACM SIGCOMM Conference*, August 2007.
- [55] Mario Čagalj, Jean-Pierre Hubaux, and Christian Enz. Minimum-energy broadcast in all-wireless networks: Np-completeness and distribution issues. In *ACM Mobicom*, pages 172–182, 2002.
- [56] W.E. Walsh, G. Tesauro, J.O. Kephart, and R. Das. Utility functions in autonomic systems. In *Autonomic Computing*, pages 70–77, 2004.
- [57] Feng Wang, Yongqiang Xiong, and Jiangchuan Liu. mtreebone: A collaborative tree-mesh overlay network for multicast video streaming. *Parallel and Distributed Systems, IEEE Transactions on*, 21(3):379–392, March 2010.
- [58] R. Wattenhofer and A. Zollinger. XTC: a practical topology control algorithm for ad-hoc networks. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 216–223, 2004.
- [59] M. Wichtlhuber, M. Mousavi, H. Al-Shatri, A. Klein, and D. Hausheer. Towards a framework for cross layer incentive mechanisms for multihop video dissemination. In *16th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–3, June 2015.
- [60] M. Wichtlhuber, B. Richerzhagen, J. Ruckert, and D. Hausheer. Transit: Supporting transitions in peer-to-peer live video streaming. In *Networking Conference, 2014 IFIP*, pages 1–9, June 2014.
- [61] Hubert Zimmermann. Osi reference model—the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4):425–432, 1980.
-

## B An OSI Model & Terms Primer

The OSI standard put forth by the ISO was meant to service three purposes:

1. *unified terminology*, such as to facilitate communication about computer networks
2. *unified general model*, such as to facilitate the construction of open computer networks to which arbitrary stakeholders could contribute components that would be capable of cooperating with appropriate other components provided by other stakeholders (hence 'Open Systems Interconnection' or OSI)
3. standardization of a concrete 7-layer architecture and of concrete protocols.

Since the latter attempt was for a great part commercially unsuccessful, we will skip it entirely and turn the first and second point. In essence, the OSI model concerns two parts of a network architecture:

1. data structures such as packets ('protocol data units', PDUs) consisting of headers ('protocol control information', PCI) and payload ('service data units', SDU), etc.
2. functional elements of layered communication systems

We will only briefly recall the second aspect above, as illustrated in Figure 11.

The following terms denote the key elements of the OSI model; they have been commonly accepted as a consequence of the spread and use of the ISO OSI standard.

- **layer:** layers contain basically the 'computer network equivalent' of the virtual machine concept known from software engineering: hiding away realization details, a higher layer (layer (N+1), say) provides a more powerful abstraction of a computer network than a lower layer (layer (N), say). Since a layer also hides away lower layers, applications or higher layer services must not circumvent (leave out) any layer. It was said above that layers do not conceptualize but contain distributed virtual machines; as such, layers are mere 'hulls' for the actual functionality, namely services.
- **service:** layers may contain a single service, several comparable services (cf. TCP and UDP on the transport layer) or complementary services (cf. the IP companion services such as ICMP and ND).

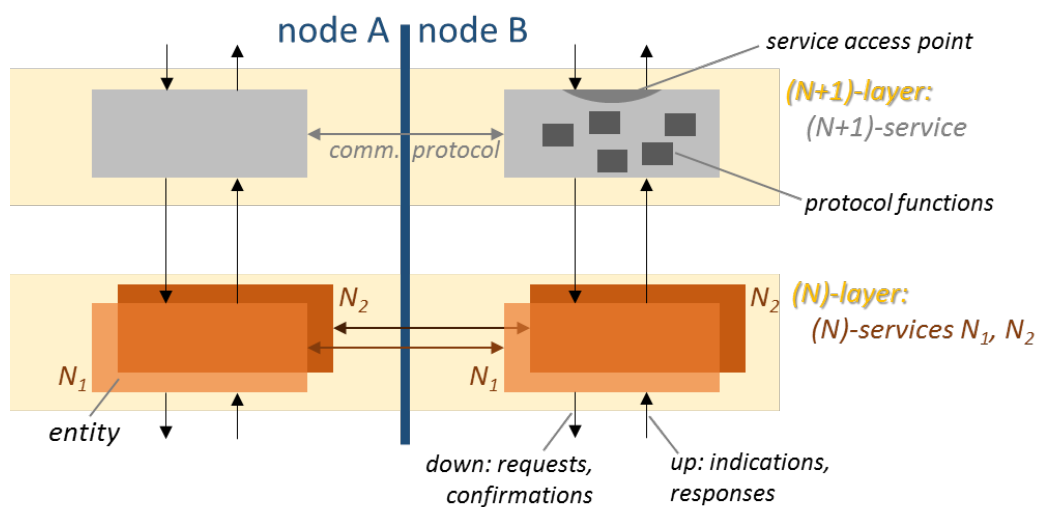


Figure 11: Functional elements of layered communication systems

- (protocol) **entity**: this term denotes the actual (usually: software) components which collectively provide a service; most services are realized as a distributed set of identical entities, some (e.g. unidirectional or strictly client server structured) services may be composed of different kinds of entities on different nodes.
- **service access point, request/indication, response/confirmation**: the functionality of a service is basically defined by means of the command/reply interface offered by that service as a means for establishing a relationship between service user and service provider: it concerns (i) the location and the addressing means for 'talking to the service', called service access point in OSI lingo (cf. TCP sockets), (ii) the commands and replies exchanged at the service access point, denoted 'requests and indications' in OSI lingo (or 'responses and confirmations', see further below); note that a 'request' at a node A—such as a request to send a message—will usually lead to an indication at a different node (B, say), such as the delivery callback event for that packet triggered at the entity of the service user on the other side; (iii) rules for how the requests, indications etc. are to be ordered, which reactions can be expected as a result of the service provision (specified as resulting indications etc.), and further details. If the functioning of a protocol requests not only a request and indication pair, but also a reaction by the addressee of the request i.e. receiver of the indication, then this second pair is denoted as response and confirmation; the standard example for this case is con-

nection establishment, where the caller issues a `connect.request`, leading to a `connect.indicate` at the callee side, and the callee—if willing to accept the call—is expected to issue a `connect.response` that leads to a `connect.confirm` on the caller side.

- (communication) **protocol**: in OSI terminology, ‘protocol’ defines the interaction among service entities and is strictly hidden from the service users i.e. higher layers (note that the P in TCP and UDP is misleading since the IETF does not strictly distinguish between service and protocol; a statement like ‘my application is using TCP’ indicates that one talks about the service, not the protocol – however, there is no such term as TCS for ‘transport control service’). OSI requires a ‘protocol’ to be defined by means of specifying (basically) the following: (i) data formats: protocol control information i.e. headers of the packet types used, minimum and maximum sizes of service data units i.e. packets, etc.); (ii) exchange and action rules: which packet type may follow which other one; how is the request/reply/response/confirm interaction at the service access point related to the exchange of protocol data units (packets) among service entities; how is this exchange of protocol data units realized by leveraging the next lower layer (and thereby: which lower layer service is assumed to be available).
- (communication) **protocol**: in OSI terminology, ‘protocol’ defines the interaction among service entities and is strictly hidden from the service users i.e. higher layers (note that the P in TCP and UDP is misleading since the IETF does not strictly distinguish between service and protocol; a statement like ‘my application is using TCP’ indicates that one talks about the service, not the protocol – however, there is no such term as TCS for ‘transport control service’). OSI requires a ‘protocol’ to be defined by means of specifying (basically) the following: (i) data formats: protocol control information i.e. headers of the packet types used, minimum and maximum sizes of service data units i.e. packets, etc.); (ii) exchange and action rules: which packet type may follow which other one; how is the request/reply/response/confirm interaction at the service access point related to the exchange of protocol data units (packets) among service entities; how is this exchange of protocol data units realized by leveraging the next lower layer (and thereby: which lower layer service is assumed to be available).
- **protocol function**: the OSI standard provides a means for structuring complex protocols into smaller functional units, so called protocol functions. However, such decomposition cannot be made freely, e.g., accord-

---

ing to software engineering principles – at least, the OSI standard does not provide noteworthy support for such ‘free’ modularization. Taken strictly, protocol functions are only foreseen for the major ‘blocks of functionality’ which are exposed to the user by means of corresponding requests and replies plus, if applicable, responses and confirmations. In the example of connection establishment cited above, ‘connect’ as in `connect.request` etc. denotes such a protocol function. A more relaxed interpretation of the OSI standard has often been applied in the past, leading to an interpretation of ‘protocol functions’ as building blocks of a communication protocol, modularized such that understanding of the functionality can be conveyed. Under this interpretation, functional aspects like addressing, packet sequence numbering, flow and congestion control etc. were denoted as protocol functions.